

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение
высшего образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

Кафедра №43

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

МЕТОДИЧЕСКОЕ ПОСОБИЕ

Санкт-Петербург 2017 г.

Оглавление

| | |
|--------------------------------|----|
| Лабораторная работа №1 | 3 |
| Лабораторная работа №2 | 8 |
| Лабораторная работа №3 | 15 |
| Лабораторная работа №4 | 17 |
| Лабораторная работа №5 | 19 |
| Лабораторная работа №6 | 23 |
| Лабораторная работа №7 | 24 |
| Лабораторная работа №8 | 27 |
| Создание проекта | 28 |
| Создание интерфейса | 31 |
| Бизнес-логика приложения | 35 |
| Лабораторная работа №9 | 40 |
| Лабораторная работа №10 | 45 |
| Приложение 1 | 48 |

Лабораторная работа №1

«Классы, конструкторы и деструкторы, одиночное наследование, права доступа»

Цель работы

Изучить принципы создания классов, наследования и ограничения прав доступа к полям и методам класса, применив на практике знания базовых синтаксических конструкций языка C++ и объектно-ориентированного программирования.

Закрепить знания по теме: Классы, конструкторы и деструкторы, одиночное наследование, права доступа.

Выбор варианта задания

Определить вариант задания, равный порядковому номеру студента в журнале (взять свой порядковый номер по модулю количества вариантов при необходимости).

Требования к лабораторной работе на желаемую оценку

Оценка «удовлетворительно»: выполнить все требования к ЛР, описанные в ее формулировке, подготовить объяснение структуры программы и принципов ее функционирования, продемонстрировать рабочую программу.

Оценка «хорошо»: ответить на 3 случайных контрольных вопроса по ЛР.

Оценка «отлично»: заменить все операторы new/delete на «умные указатели» такие, как `std::shared_ptr<T>`, `std::weak_ptr<T>` или `std::unique_ptr<T>`.

Описание работы

В работе необходимо реализовать класс в соответствии с вариантом задания и создать приложение для вызова методов класса в функции `main()`. Поля класса должны иметь спецификатор доступа `private` и `public` методы `setFieldName` и `getFieldName` для каждого поля класса. Поля должны быть объявлены, как указатели. В конструкторе класса должна быть инициализация полей, через оператор `new`, в деструкторе освобождение памяти, через оператор `delete`.

Варианты заданий

Вариант 1

Создать класс «Date» для работы с датами в формате «год.месяц.день». Дата представляется структурой с тремя полями:

- «Год» (тип `std::uint16_t`);
- «Месяц» (тип `std::uint8_t`);
- «День» (тип `std::uint8_t`).

Класс должен включать не менее трех конструкторов:

- инициализации числами;
- инициализации строкой вида «год.месяц.день» (например, «2004.08.31»);
- инициализации датой.

Должны быть реализованы методы:

- вычисление даты через заданное количество дней;
- вычитание заданного количества дней из даты;
- определение високосного года;
- присвоение и получение отдельных частей (год, месяц, день);
- сравнение дат (равно, до, после);
- вычисление количества дней между датами.

Вариант 2

Создать класс «LongLong» для работы с целыми числами из 128 бит. Число должно быть представлено двумя полями:

- Старшая часть (тип `std::int64_t`);
- Младшая часть (тип `std::uint64_t`).

Должны быть реализованы методы:

- Сложение (функция `add`);
- Вычитание (функция `sub`);
- Умножение (функция `mul`);
- Деление (функция `div`);
- Остаток от деления (функция `mod`).

Вариант 3

Создать класс «BitString» для работы с 128-битовыми строками. Битовая строка должна быть представлена двумя полями типа `std::uint64_t`. Должны быть реализованы все традиционные методы для работы с битами:

- Побитовое «И» (функция `and`);
- Побитовое «ИЛИ» (функция `or`);
- Исключающее «ИЛИ» (функция `xor`);
- Побитовое отрицание (функция `not`).
- Должны быть реализованы методы смещения на заданное количество бит:
- Сдвиг влево (функция `shiftLeft`);
- Сдвиг вправо (функция `shiftRight`).

Вариант 4

Создать класс «Rational» для работы с рациональными дробями. Рациональная (несократимая) дробь представляется парой целых чисел (a, b). Где, a — числитель, b — знаменатель. Должны быть реализованы методы:

- Сложение add: $(a, b) + (c, c!) = (ad + bc, bd)$;
- Вычитание sub: $(a, b) - (c, d) = (ad - bc, bd)$;
- Умножение mul: $(a, b) \times (c, c!) = (ac, bd)$;
- Деление div: $(a, b) / (c, c!) = (ad, bc)$;
- Сравнение: eq (equal), gt (greater then), lt (lower then).

Обязательно должна быть реализована приватная функция сокращения дроби reduce, которая обязательно вызывается при выполнении арифметических операций.

Вариант 5

Создать класс «Vectors» для объектов-векторов, задаваемых координатами концов в трехмерном пространстве. Должны быть реализованы методы:

- Сложения и вычитания векторов с получением нового вектора (суммы или разности);
- Вычисления скалярного произведения двух векторов, длины вектора, косинуса угла между векторами.

Вариант 6

Создать класс «BinaryTree», реализующий бинарное дерево.

- Должны быть реализованы методы:
- Добавление элемента (функция add);
- Удаление элемента (функция del);
- Поиск элемента по ключу (функция search);
- Последовательного доступа ко всем элементам (функция get).

Вариант 7

Создать класс «ComplexNumber» для представления комплексных чисел. Должны быть реализованы методы:

- Сложение (функция add);
- Вычитание (функция sub);
- Умножение (функция mul).

Вариант 8

Создать класс «Vector» для определения одномерных массивов целых чисел (векторов). Должны быть реализованы методы:

- Обращения к отдельному элементу массива с контролем выхода за пределы массива;
- Возможность задания произвольных границ индексов при создании объекта; Возможность выполнения операций поэлементного сложения и вычитания массивов с одинаковыми границами индексов;
- Умножения и деления всех элементов массива на скаляр;

Вариант 9

Создать класс «StringMass» для определения одномерных массивов строк фиксированной длины.

Должны быть реализованы методы:

- Обращения к отдельным строкам массива по индексам, контроль выхода за пределы массива;
- Поэлементного сцепление двух массивов с образованием нового массива;
- Слияния двух массивов с исключением повторяющихся элементов;

Вариант 10

Создать класс «Polynomial» для определения многочленов от одной переменной, задаваемых степенью многочлена и массивом коэффициентов.

Должны быть реализованы методы:

- Вычисления значения многочлена для заданного аргумента;
- Операции сложения, вычитания и умножения многочленов с получением нового объекта-многочлена;

Вариант 11

Создать класс «Matrix» для определения матрицы произвольного размера.

Должны быть реализованы методы:

- Изменения числа строк и столбцов;
- Умножение, сложение, вычитание, сравнение матриц;
- Проверка типа матрицы (квадратная, диагональная, нулевая, единичная, симметричная, верхняя треугольная, нижняя треугольная).

Вариант 12

Создать класс «Set» для определения множества.

Должны быть реализованы методы:

- Добавление элемента;
- Удаление элемента;
- Пересечение, объединение и разность множеств.

Контрольные вопросы

1. Что такое конструктор и в чем его назначение? Может ли существовать несколько конструкторов, если да, показать, чем они будут различаться. Привести пример.
2. Что такое деструктор и в чем его назначение? Может ли существовать несколько деструкторов, если да, показать, чем они будут различаться. Привести пример.
3. Что такое конструктор по умолчанию? Привести пример.
4. Что такое деструктор по умолчанию? Привести пример.
5. Зачем нужны спецификаторы доступа `private`, `protected` и `public`? Какое удобство они дают?
6. Чем описание класса отличается от экземпляра класса? Привести пример.
7. Что такое экземпляр класса? Привести пример.
8. Может ли один и тот же класс иметь несколько описаний? Если да, привести пример.
9. Может ли один и тот же класс иметь несколько экземпляров? Если да, привести пример.
10. Что называется методом, а что полем класса? Привести пример.
11. Как получить доступ к полю класса или его методу? Привести пример.
12. В чем отличие статического и не статического полей класса? Методов?
13. Влияет ли на размер класса количество его методов?
14. Что называют «интерфейсом» класса? Привести пример.
15. Что такое статический класс? В чем его преимущества и недостатки?

Лабораторная работа №2

«Виртуальные функции и абстрактные классы»

Цель работы

Изучить принципы построения консольных приложений, применив на практике знания базовых синтаксических конструкций языка C++ и объектно-ориентированного программирования.

Закрепить знания по теме: Виртуальные функции и абстрактные классы.

Выбор варианта задания

Определить вариант задания, равный порядковому номеру студента в журнале (взять свой порядковый номер по модулю количества вариантов при необходимости).

Требования к лабораторной работе на желаемую оценку

Оценка «удовлетворительно»: выполнить все требования к ЛР, описанные в ее формулировке, подготовить объяснение структуры программы и принципов ее функционирования, продемонстрировать рабочую программу.

Оценка «хорошо»: ответить на 3 случайных контрольных вопроса по ЛР.

Оценка «отлично»: изменить положение одной фигуры на выбор.

Описание работы

Производные классы — это простое, гибкое и эффективное средство определения класса с целью повторного использования готового программного кода. Новые возможности добавляются к уже существующему классу, не требуя его перепрограммирования или перекомпиляции. С помощью производных классов можно организовать общий интерфейс с несколькими различными классами так, что в других частях программы можно будет единообразно работать с объектами этих классов. Понятие виртуальной функции позволяет использовать объекты надлежащим образом даже в тех случаях, когда их тип на стадии трансляции неизвестен. Основное назначение производных классов — упростить программисту задачу выражения общности классов.

Любое понятие не существует изолированно, оно существует во взаимосвязи с другими понятиями, и мощность данного понятия во многом определяется наличием таких связей. Раз класс служит для представления понятий, встаёт вопрос, как представить взаимосвязь понятий. Понятие производного класса и поддерживающие его языковые средства служат для представления иерархических связей, иными словами, для выражения общности между классами. Например, понятия окружности и треугольника связаны между собой, так как оба они представляют ещё понятие фигуры, т. е. содержат более общее понятие. Чтобы представлять в программе окружности и треугольники и при этом не упускать из вида, что они являются фигурами, надо явно определять классы окружность и треугольник так, чтобы было видно, что у них есть общий класс — фигура. Эта простая идея по сути является основой того, что обычно называется объектно-ориентированным программированием.

Рассмотрим учебную программу (

Приложение 1), использующую некоторые из этих идей. Программа предназначена для вывода на экран картинок, составленных из набора заготовок — «фигур». В программе объявлен абстрактный класс «фигура» (*Shape*). Все конкретные фигуры — линия, прямоугольник и т. п. — являются производными от этого класса. Класс «фигура» поддерживает действия, необходимые для всех фигур: он создаёт из всех объявляемых фигур общий список, который может быть обработан программой рисования при выдаче фигур на экран. Кроме того, в классе «фигура» объявлен набор функций-членов, которые должны поддерживать все фигуры, чтобы из них можно было создавать картинки. Это функции, возвращающие координаты всех крайних точек фигуры, по которым их можно будет стыковать. Эти функции — чисто виртуальные, они должны быть обязательно определены затем отдельно в каждой фигуре. Имеются также два дополнительных класса, уточняющие свойства фигур. Некоторые фигуры можно поворачивать. Для таких фигур имеется базовый класс *Rotatable*. Для других фигур возможна операция отражения относительно горизонтальной или вертикальной оси. Эти фигуры можно строить на базе класса *Reflectable*. Если фигура имеет оба этих свойства, то она должна быть производной от обоих классов.

Класс «фигура» является ядром библиотеки фигур *shape.hpp*. Имеется также библиотека поддержки работы с экраном *screen.hpp*, в которой определены размеры экрана, введено понятие точки и перечислены утилиты работы с экраном, конкретизированные затем в *shape.hpp*. Для простоты и универсальности работа с экраном реализована как формирование и построчный вывод матрицы символов.

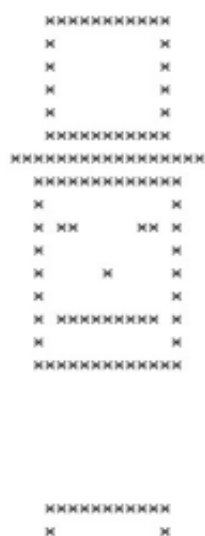
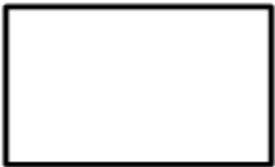



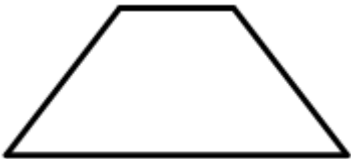
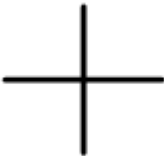


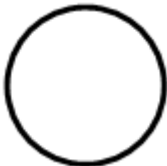


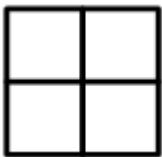






Рис. 1. Результат работы программы

В таблице 1 собрана коллекция фигур, которыми можно дополнить рассмотренную прикладную программу.

Таблица 1. Коллекция дополнительных фигур

| № п/п | Наименование | Вид | Отражение | Поворот |
|----------|--------------|-----|-----------|---------|
| | | | | |

| | | | | |
|---|----------------|--|-----|-----|
| 1 | Прямоугольник |  | Нет | Да |
| 2 | Квадрат |  | Нет | Нет |
| 3 | Ромб |  | Нет | Нет |
| 4 | Параллелограмм |  | Да | Да |
| 5 | Трапеция |  | Да | Да |
| 6 | Крест |  | Нет | Нет |
| 7 | Косой крест |  | Нет | Нет |
| 8 | Треугольник |  | Да | Да |
| 9 | Кружок |  | Нет | Нет |

| | | | | |
|----|-------------------------|---|-----|-----|
| 10 | Квадрат с крестом |  | Нет | Нет |
| 11 | Ромб с крестом |  | Нет | Нет |
| 12 | Кружок с крестом |  | Нет | Нет |
| 13 | Треугольник с крестом |  | Да | Да |
| 14 | Зачёркнутый квадрат |  | Нет | Нет |
| 15 | Зачёркнутый кружок |  | Нет | Нет |
| 16 | Зачёркнутый треугольник |  | Да | Да |

Для некоторых фигур возможны поворот на 90° вправо или влево, или отражение относительно горизонтальной и/или вертикальной оси симметрии, причём для части из них имеются обе возможности. Некоторые фигуры строятся как составные из более простых. Эти идеи можно отобразить показанной на рис. 2 иерархией классов.

Доработать учебную программу: добавить в коллекцию ещё одну фигуру, номер которой указан в таблице 1. Для этой фигуры нужно будет определить подходящее место в иерархии классов и написать необходимые функции-члены. Функции-члены, использование которых не предполагается, можно определить так, чтобы они были недоступны. Разработанной фигурой нужно дополнить картинку в указанных в варианте позициях. Позиция 1 обозначает галстук или воротник, 2 и 3 — бакенбарды, 4 и 5 — уши, 6 — кокарду, 7 и 8 — рога, 9 — нос, 10 и 11 — глаза, 12 — шляпу в целом (см. Рис. 3). Возможно, некоторые из фигур нужно будет повернуть или отразить. Для примыкания фигур должны использоваться их габаритные точки.

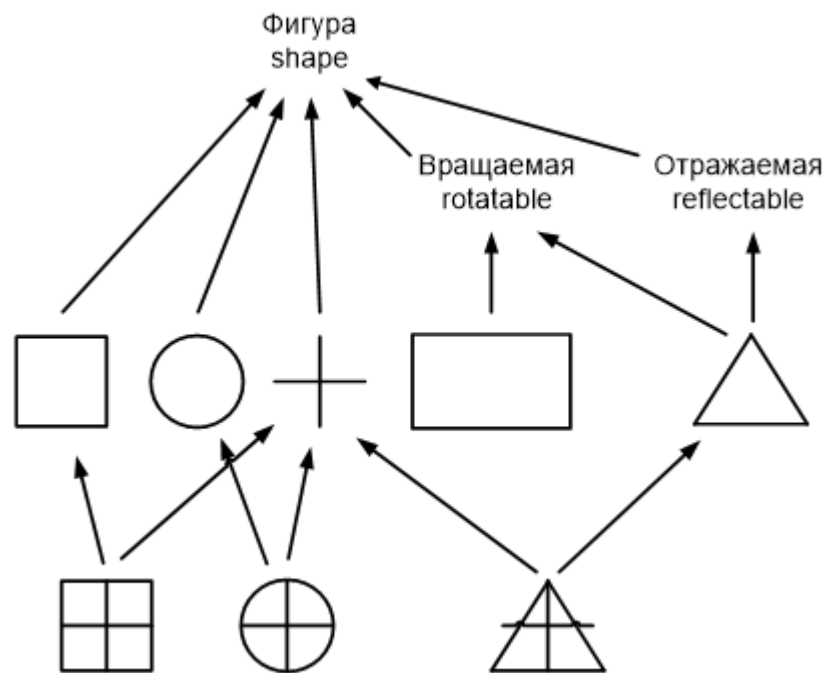


Рис 2. Фрагмент иерархии классов фигур

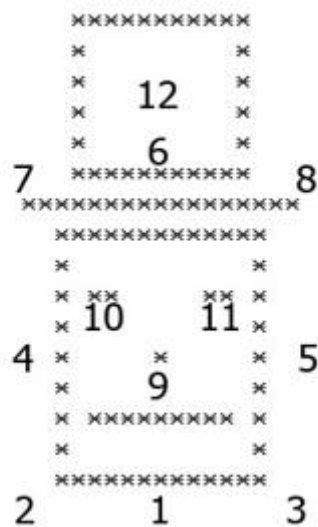


Рис 3. Позиции на результате работы программы для возможной вставки дополнительной фигуры

Варианты заданий

Таблица 2. Индивидуальные задания по лабораторной работе

| № варианта | Фигура | Расположение | № варианта | Фигура | Расположение |
|------------|--------|--------------|------------|--------|---------------|
| 1 | 2 | 2, 3, 10, 11 | 26 | 15 | 6, 7, 8 |
| 2 | 7 | 1, 4, 5 | 27 | 12 | 9, 10 11 |
| 3 | 6 | 4, 5 | 28 | 11 | 7, 8, 12 |
| 4 | 3 | 10, 11, 12 | 29 | 2 | 4, 5, 9 |
| 5 | 8 | 4, 5, 6 | 30 | 11 | 1, 7, 8 |
| 6 | 5 | 1 | 31 | 12 | 1, 2, 3 |
| 7 | 4 | 7, 8 | 32 | 3 | 1, 4, 5 |
| 8 | 9 | 10, 11, 12 | 33 | 10 | 4, 5 |
| 9 | 4 | 2, 3 | 34 | 13 | 1, 7, 8 |
| 10 | 5 | 4, 5 | 35 | 4 | 7, 8 |
| 11 | 10 | 1, 12 | 36 | 9 | 1, 4, 5 |
| 12 | 3 | 4, 5, 9 | 37 | 14 | 4, 5, 6 |
| 13 | 6 | 1, 12 | 38 | 5 | 4, 5 |
| 14 | 11 | 10, 11 | 39 | 8 | 1, 6, 9 |
| 15 | 2 | 1, 7, 8, 12 | 40 | 15 | 1, 2, 3 |
| 16 | 7 | 6, 7, 8 | 41 | 6 | 2, 3, 9 |
| 17 | 12 | 2, 3, 7, 8 | 42 | 7 | 10, 11 |
| 18 | 15 | 10, 11 | 43 | 2 | 1, 2, 3, 7, 8 |
| 19 | 8 | 2, 3, 7, 8 | 44 | 7 | 2, 3, 12 |
| 20 | 13 | 1, 12 | 45 | 6 | 10, 11 |
| 21 | 14 | 2, 3, 12 | 46 | 3 | 2, 3, 6 |
| 22 | 9 | 2, 3, 9 | 47 | 8 | 9, 10, 11 |
| 23 | 14 | 7, 8, 9 | 48 | 5 | 12 |
| 24 | 13 | 1, 4, 5 | 49 | 4 | 4, 5 |
| 25 | 10 | 7, 8 | 50 | 9 | 4, 5, 7, 8 |

Контрольные вопросы

1. Какие преимущества дает наследование?
2. Как влияют спецификаторы доступа на поля и методы базового класса?
3. Что такое абстрактный класс? Привести пример.
4. Что такое виртуальная функция и зачем она нужна? Привести пример.
5. Что такое виртуальный деструктор и зачем он нужен? Привести пример.
6. Как запретить наследование от класса? Привести пример.
7. Зачем нужны преобразования указателей/ссылок от производных классов к базовому?
8. Что такое виртуальное наследование? Проблема ромбовидного наследования.
9. Как вызвать определенный конструктор базового класса?
10. Можно ли обратиться к одноименной функции базового класса из производного?
11. В каком порядке создаются и разрушаются унаследованные объекты и их базовые классы?
12. Что такое чисто виртуальная функция? Привести пример.
13. Зачем нужен спецификатор `override` в C++ ?
14. Что такое таблица виртуальных функций? Привести пример таблицы.
15. Можно ли при помощи наследования вызывать закрытый метод производного класса?

Лабораторная работа №3

«Операторные функции в пространстве имен и как члены класса»

Цель работы

Изучить принципы построения приложений, применив на практике знания базовых синтаксических конструкций языка C++ и объектно-ориентированного программирования.

Закрепить знания по теме: Операторные функции в пространстве имен и как члены класса.

Выбор варианта задания

В соответствии с вариантом выбранным в лабораторной работе №1.

Требования к лабораторной работе на желаемую оценку

Оценка «удовлетворительно»: выполнить все требования к ЛР, описанные в ее формулировке, подготовить объяснение структуры программы и принципов ее функционирования, продемонстрировать рабочую программу.

Оценка «хорошо»: ответить на вопросы, определяются при сдаче ЛР.

Оценка «отлично»: Индивидуальное задание, определяется при сдаче ЛР.

Описание работы

В работе необходимо дополнить класс реализованный в лабораторной работе №1, операторными функциями, используя перегрузку операторов. Реализовать перегрузку трёх операторов по заданию.

C++ поддерживает перегрузку операторов (operator overloading). За небольшими исключениями большинство операторов C++ могут быть перегружены, в результате чего они получают специальное значение по отношению к определенным классам. Например, класс, определяющий связанный список, может использовать оператор + для того, чтобы добавлять объект к списку. Другой класс может использовать оператор + совершенно иным способом. Когда оператор перегружен, ни одно из его исходных значений не теряет смысла. Просто для определенного класса объектов определен новый оператор. Поэтому перегрузка оператора + для того, чтобы обрабатывать связанный список, не изменяет его действия по отношению к целым числам. Для того, чтобы перегрузить оператор, необходимо определить, что именно означает оператор по отношению к тому классу, к которому он применяется. Для этого определяется функция-оператор, задающая действие оператора. Общая форма записи функции-оператора для случая, когда она является членом класса, имеет вид:

```
тип имя_класса::operator#(список_аргументов)
{
    // действия, определенные применительно к классу
}
```

Здесь перегруженный оператор подставляется вместо символа #, а тип задает тип значений, воз-вращаемых оператором. Для того, чтобы упростить использование перегруженного оператора в сложных выражениях, в качестве возвращаемого значения часто выбирают тот же самый тип, что и класс, для которого перегружается оператор.

Варианты заданий

| | | | |
|-------------------|------------------|-----------------------|-----------------------------|
| Вариант 1 | Присваивание (=) | Равенство (==) | Сложение (+) |
| Вариант 2 | Присваивание (=) | Больше (>) | Вычитание (-) |
| Вариант 3 | Присваивание (=) | Меньше (<) | Побитовое «И» (&) |
| Вариант 4 | Присваивание (=) | Больше или равно (>=) | Умножение (*) |
| Вариант 5 | Присваивание (=) | Меньше или равно (<=) | Сложение (+) |
| Вариант 6 | Присваивание (=) | Равенство (==) | Обращение к элементу (a[b]) |
| Вариант 7 | Присваивание (=) | Больше (>) | Умножение (*) |
| Вариант 8 | Присваивание (=) | Меньше (<) | Обращение к элементу (a[b]) |
| Вариант 9 | Присваивание (=) | Больше или равно (>=) | Обращение к элементу (a[b]) |
| Вариант 10 | Присваивание (=) | Меньше или равно (<=) | Сложение (+) |
| Вариант 11 | Присваивание (=) | Равенство (==) | Вычитание (-) |
| Вариант 12 | Присваивание (=) | Больше (>) | Вычитание (-) |

Лабораторная работа №4

«Шаблоны классов»

Цель работы

Изучить принципы построения консольных приложений, применив на практике знания базовых синтаксических конструкций языка C++ и объектно-ориентированного программирования.

Закрепить знания по теме: Шаблоны классов.

Выбор варианта задания

Определить вариант задания, равный порядковому номеру студента в журнале (взять свой порядковый номер по модулю количества вариантов при необходимости).

Требования к лабораторной работе на желаемую оценку

Оценка «удовлетворительно»: выполнить все требования к ЛР, описанные в ее формулировке, подготовить объяснение структуры программы и принципов ее функционирования, продемонстрировать рабочую программу.

Оценка «хорошо»: ответить на вопросы, определяются при сдаче ЛР.

Оценка «отлично»: Индивидуальное задание, определяется при сдаче ЛР.

Описание работы

В работе необходимо реализовать класс в соответствии с вариантом задания и создать приложение для вызова методов класса в функции main(). Реализуемый класс должен представлять из себя структуру данных и уметь работать с встроенными типами данных, таким как: int, double, float, char. И реализовывать функции вставки, удачления и в зависимости от варианта доступ по ключу или индексу.

Варианты заданий

| Вариант | Название класса | Описание |
|------------|-----------------|--|
| Вариант 1 | Array | Статический непрерывный массив |
| Вариант 2 | Vector | Динамический непрерывный массив |
| Вариант 3 | Deque | Двусторонняя очередь |
| Вариант 4 | ForwardList | Односвязный список |
| Вариант 5 | List | Двусвязный список |
| Вариант 6 | Set | Коллекция уникальных ключей, отсортированная по ключам |
| Вариант 7 | Map | Коллекция пар ключ-значение, отсортированная по ключам, ключи являются уникальными |
| Вариант 8 | Stack | Адаптируется контейнеров обеспечить стек (LIFO структуры данных) |
| Вариант 9 | Queue | Адаптируется контейнера обеспечивают очереди (FIFO структуры данных) |
| Вариант 10 | PriorityQueue | Адаптация контейнеров для обеспечения приоритета очереди |

Лабораторная работа №5

«Обработка исключительных ситуаций»

Цель работы

Изучить принципы построения консольных приложений, применив на практике знания базовых синтаксических конструкций языка C++ и объектно-ориентированного программирования.

Закрепить знания по теме: Обработка исключительных ситуаций.

Требования к лабораторной работе на желаемую оценку:

Оценка «удовлетворительно»: выполнить все требования к ЛР, описанные в ее формулировке, подготовить объяснение структуры программы и принципов ее функционирования, продемонстрировать рабочую программу;

Оценка «хорошо»: ответить на 3 случайных контрольных вопроса по ЛР;

Оценка «отлично»: обосновать набор и вид классов для фиксации особых ситуаций, место расположения операторов `throw` и блоков контроля с целью получения безопасного кода.

Описание работы

Механизм исключительных ситуаций предоставляет пользователю возможность контроля за ходом выполнения программы и механизм нейтрализации возможных ошибок. Главное в этом механизме — поддержка действий по нейтрализации последствий события, препятствующего продолжению нормальной работы. Очень часто исключительная ситуация — это не ошибка, а просто исчерпание какого-то ресурса, например, доступной памяти или времени ожидания сигнала или даже один из предусмотренных вариантов завершения процесса. Механизм используется, если ситуация не может быть разрешена в той точке, где была выявлена, и требует перехода на более высокий уровень, с завершением каких-то активных функций и освобождением ресурсов.

Для задействования механизма особых ситуаций в программе нужно проделать следующее:

— обнаружив в программе место, где особая ситуация может возникнуть, надо придумать для неё уникальное название, например *My_Error*, и объявить соответствующий класс ошибок, возможно, пустой:

```
class My_Error { };
```

— в точке программы, в которой обнаружена особая ситуация, поместить утверждение `throw My_Error();`

Это утверждение создаёт объект класса *My_Error*. Если в объекте предусмотрены поля для данных, через них можно передать информацию обработчику ошибок: аргумент утверждения *throw* — это конструктор объекта;

— точку вызова функции, которая может создать исключения, нужно поместить в блок

контроля, за которым следуют обработчики особых ситуаций:

```
try{ //начало блока контроля;  
    вызов функции, которая может создать особую ситуацию  
    (содержит утверждения throw My_Error);  
}  
catch (My_Error)  
{ обработка особой ситуации }
```

Предложение *catch* размещается на том уровне, где обработка ситуации *My_Error* возможна. Если нужно обрабатывать несколько различных ошибок, после блока *try* последовательно размещаются соответствующие обработчики. При возникновении любой особой ситуации в блоке *try* его работа прерывается: происходит принудительный выход из всех функций, которые были активны в точке особой ситуации, и вызов деструкторов для всех созданных при этом объектов, как это происходит при выходе из блока (области видимости). Этот процесс называется раскруткой стека: стек возвращается в состояние, в котором он был при входе в блок *try*.

Далее просматриваются блоки *catch* в том порядке, в каком они объявлены. Как только обнаруживается блок обработки ошибок нужного типа, управление передаётся ему. Остальные блоки *catch* не используются.

Если же выполнение блока *try* завершилось успешно, все блоки *catch* после него игнорируются.

Если для некоторого типа ошибки не обнаружено соответствующего блока *catch*, программа завершается аварийно. Чтобы этого избежать, последним в цепочке можно разместить блок *catch(...)*, перехватывающий ошибки любого типа.

Как только подходящий блок *catch* будет вызван, особая ситуация будет считаться обработанной, даже если этот блок пуст. Однако чаще всего в него помещают выдачу на экран или в специальный файл (журнал) содержательного сообщения об ошибке. Возможно также одно из следующих действий:

- устранение причины ошибки (уменьшение запроса на выделение памяти, отказ от обработки несуществующего или испорченного файла и т. п.);
- аварийное завершение программы (вызов *abort()*);
- перевозбуждение особой ситуации для передачи на следующий уровень иерархии (вызов *throw* без аргумента).

Правильный выбор уровня для размещения блока контроля позволяет сделать программу безопасной в смысле исключений. Так, в лабораторной работе №3 имеется следующая цепочка вызовов функций:

```
main( ) → MyShape::draw( screen ) → Square::draw ( screen ) → Line:: draw ( screen ) →
```

`Screen::putLine(a, b) → Screen::putLine(x0, y0, x1, y1) → putPoint(x0, y0).`

Выход точки за пределы буферного массива `_screen` (экрана) выявляется функцией `putPoint(...)`. Блок контроля вокруг вызова этой функции (или вызывающей её `putPoint`) не имеет смысла: на этом уровне ничего, кроме выдачи сообщения об ошибке, сделать нельзя, а такое сообщение можно выдать непосредственно, не прибегая к механизму `throw — catch`. На уровне `main()` или `MyShape::draw(...)` обрабатывать ошибку поздно, можно только прервать выполнение программы, содержательное сообщение о месте ошибки получить нельзя. В то же время блок контроля внутри функции `Square::draw(...)` позволит локализовать ошибку при выводе прямоугольника — контура фигуры `MyShape` и, возможно, попробовать изменить его размер. В общем случае проектирование реакции программной системы на ошибки должно выполняться одновременно с проектированием её самой. Так, в программе, рассмотренной в лабораторной работе №3, можно снабдить каждую фигуру автоматически формируемым порядковым номером, значение которого можно выводить как часть сообщения об ошибке «выход за пределы экрана».

Если ошибка выявлена в конструкторе фигуры, фигура не создаётся. Для правильной обработки такой ситуации нужно дополнить библиотеку деструктором для класса `Shape`. Вариант: заменить ошибочную фигуру специальным значком «ошибка», выводимым на картинку.

Контрольные вопросы

1. Что такое исключительная ситуация при выполнении программы?
2. Как можно выявить исключительную ситуацию?
3. Что можно предпринять в случае, когда исключительная ситуация выявлена?
4. Могут ли в одной программе появляться исключительные ситуации разных типов и если «да», то как их можно различить?
5. В каком месте программы следует поместить обработчик особых ситуаций?
6. Как можно вызвать обработчик особых ситуаций?
7. Можно ли передать в обработчик особых ситуаций какую-либо информацию о произошедшем событии?
8. Можно ли обработать неизвестную особую ситуацию?
9. Можно ли сделать обработчик ситуации пустым?
10. Что можно предпринять, если выяснилось, что для корректной обработки ситуации в данном месте программы у обработчика оказалось недостаточно данных?
11. Если требуется несколько обработчиков особых ситуаций, в каком порядке их следует размещать в программе?
12. Как действуют обработчики в случае, когда никакой особой ситуации не произошло?
13. Как следует размещать блоки контроля, чтобы получить безопасный программный код?

Лабораторная работа №6

«Стандартная библиотека C++. Библиотека ввода-вывода»

Цель работы

Изучить принципы построения консольных приложений, применив на практике знания базовых синтаксических конструкций языка C++ и объектно-ориентированного программирования.

Закрепить знания по теме: Стандартная библиотека C++. Библиотека ввода-вывода.

Выбор варианта задания

В соответствии с вариантом выбранным в лабораторной работе №1.

Требования к лабораторной работе на желаемую оценку

Оценка «удовлетворительно»: выполнить все требования к ЛР, описанные в ее формулировке, подготовить объяснение структуры программы и принципов ее функционирования, продемонстрировать рабочую программу.

Оценка «хорошо»: ответить на вопросы, определяются при сдаче ЛР.

Оценка «отлично»: Индивидуальное задание, определяется при сдаче ЛР.

Описание работы

В работе необходимо дополнить приложение реализованное в лабораторной работе №1. Добавить работу с консолью. Приложение должно осуществлять ввод и вывод информации о реализованном классе. Работа с приложением должна быть реализована в диалоговом режиме («запрос-ответ»). В качестве запроса может выступать ввод команды, либо выбор пункта меню, выведенного приложением в консоль. После получения запроса программа может потребовать ввода необходимых для выполнения запрошенной операции данных. После получения необходимой информации программа осуществляет соответствующее действие, выводит результат работы в консоль и ожидает следующего запроса.

Лабораторная работа №7

«Стандартная библиотека C++. Последовательные и ассоциативные контейнеры.
Обобщенные алгоритмы»

Цель работы

Изучить принципы построения консольных приложений, применив на практике знания базовых синтаксических конструкций языка C++ и объектно-ориентированного программирования.

Закрепить знания по теме: Классы, конструкторы и деструкторы, права доступа. Операторные функции в пространстве имен и как члены класса. Стандартная библиотека C++. Библиотека ввода-вывода.

Выбор варианта задания

Определить вариант задания, равный порядковому номеру студента в журнале (взять свой порядковый номер по модулю количества вариантов при необходимости).

Требования к лабораторной работе на желаемую оценку

Оценка «удовлетворительно»: выполнить все требования к ЛР, описанные в ее формулировке, подготовить объяснение структуры программы и принципов ее функционирования, продемонстрировать рабочую программу.

Оценка «хорошо»: ответить на вопросы, определяются при сдаче ЛР.

Оценка «отлично»: Индивидуальное задание, определяется при сдаче ЛР.

Описание работы

Приложение должно осуществлять ввод и вывод информации о реализованном классе. Заполнить массив данных (vector) случайными числами в диапазоне $m1 - m2$. Выполнить набор действий при помощи обобщенных алгоритмов, объектов-функций и предикатов. После выполнения каждого действия выводить на экран результат.

Варианты заданий

Вариант 1.

- $m1=-10, m2=10$.
- Найти максимальный элемент массива по абсолютному значению
- Заменить все элементы, значения которых меньше 0 на 0.
- Удалить из массива все повторяющиеся значения, кроме первого.

Вариант 2.

- $m1=-10, m2=10$
- найти максимальный элемент массива
- прибавить к каждому элементу массива найденный максимальный элемент
- отсортировать массив по абсолютному значению

Вариант 3.

- $m1=0, m2=10$
- вычислить среднее значение всех элементов массива
- создать второй массив такой же размерности и заполнить его случайными числами в диапазоне $m1=-10, m2=10$
- сложить поэлементно два массива
- найти количество элементов, абсолютное значение которых равно 5

Вариант 4.

- $m1=0, m2=15$
- создать второй массив как копию первого
- перемешать случайным образом все элементы 2-го массива
- вычислить среднее значение всех элементов массива
- перемножить поэлементно два массива
- возвести в квадрат каждый элемент полученного массива

Вариант 5.

- $m1=-10, m2=10$
- умножить все элементы массива на 2
- заменить все элементы, абсолютное значение которых больше 10 на 0
- удалить из массива все значения, равные 0

Вариант 6.

- $m1=-50, m2=50$
- поделить все элементы массива на 2
- обменять зеркально 2 первых элемента массива с 2 последними
- заменить все элементы, абсолютное значение которых меньше 10 на 0

Вариант 7.

- $m1=0, m2=100$
- подсчитать количество элементов со значениями больше 10
- найти корень квадратный из всех элементов
- вычислить сумму всех элементов

Вариант 8

- $m1=-10, m2=10$

- переставить все элементы массива в обратном порядке
- для каждого элемента вычислить выражение $x = (x < 0) ? 0 : x * 2$
- подсчитать количество элементов со значениями не равными 0

Вариант 9.

- $m1 = -5, m2 = 5$
- сдвинуть все элементы массива вправо на 2 элемента (циклический сдвиг)
- умножить все элементы массива на 10
- подсчитать количество элементов, квадрат которых больше p ($p = 10$)

Вариант 10.

- $m1 = 0, m2 = 10$
- вычислить сумму всех элементов
- для каждого элемента вычислить выражение $x = k * x + b$ ($k = 2, b = 5$)
- заменить все элементы большие 20 на 10

Лабораторная работа №8

«Структурные шаблоны проектирования»

Цель работы

Изучить принципы построения приложений с графическим интерфейсом, используя библиотеку Qt, применив на практике знания базовых синтаксических конструкций языка C++ и объектно-ориентированного программирования.

Закрепить знания по теме: Структурные шаблоны проектирования.

Требования к лабораторной работе:

Выполнить все требования к лабораторной работе, описанные в ее формулировке, подготовить объяснение структуры программы и принципов ее функционирования, продемонстрировать рабочую программу. Подготовить отчет в соответствии с шаблоном.

Описание работы

Шаблон проектирования, паттерн (англ. design pattern) — это многократно применяемая архитектурная конструкция, предоставляющая решение общей проблемы проектирования в рамках конкретного контекста и описывающая значимость этого решения. Паттерн не является законченным образцом проекта, который может быть прямо преобразован в код.

Это описание или образец для того, как решить задачу таким образом, чтобы это можно было использовать в различных ситуациях. Объектно-ориентированные шаблоны зачастую показывают отношения и взаимодействия между классами или объектами, без определения того, какие конечные классы или объекты приложения будут использоваться.

Основными типами шаблонов проектирования являются:

- порождающие шаблоны;
- структурные шаблоны;
- поведенческие шаблоны.

Приложение, создаваемое в лабораторной работе, предназначено для знакомства с шаблоном проектирования Фасад. Программа имитирует расчет страховых взносов. Пользователь вводит на форме данные страховщика, запускает расчет страховых взносов, а также имеет возможность вернуться к прежним запросам. Лабораторные работы № 8 – 10 предполагают последовательное выполнение и получение в конце рабочего экземпляра программы.

Информация о паттернах проектирования

1. <http://cpp-reference.ru/patterns/>
2. <http://citforum.ru/SE/project/pattern/#anno>

Создание проекта

Создайте новый проект, выбрав в качестве типа проекта «Приложение Qt Widgets».

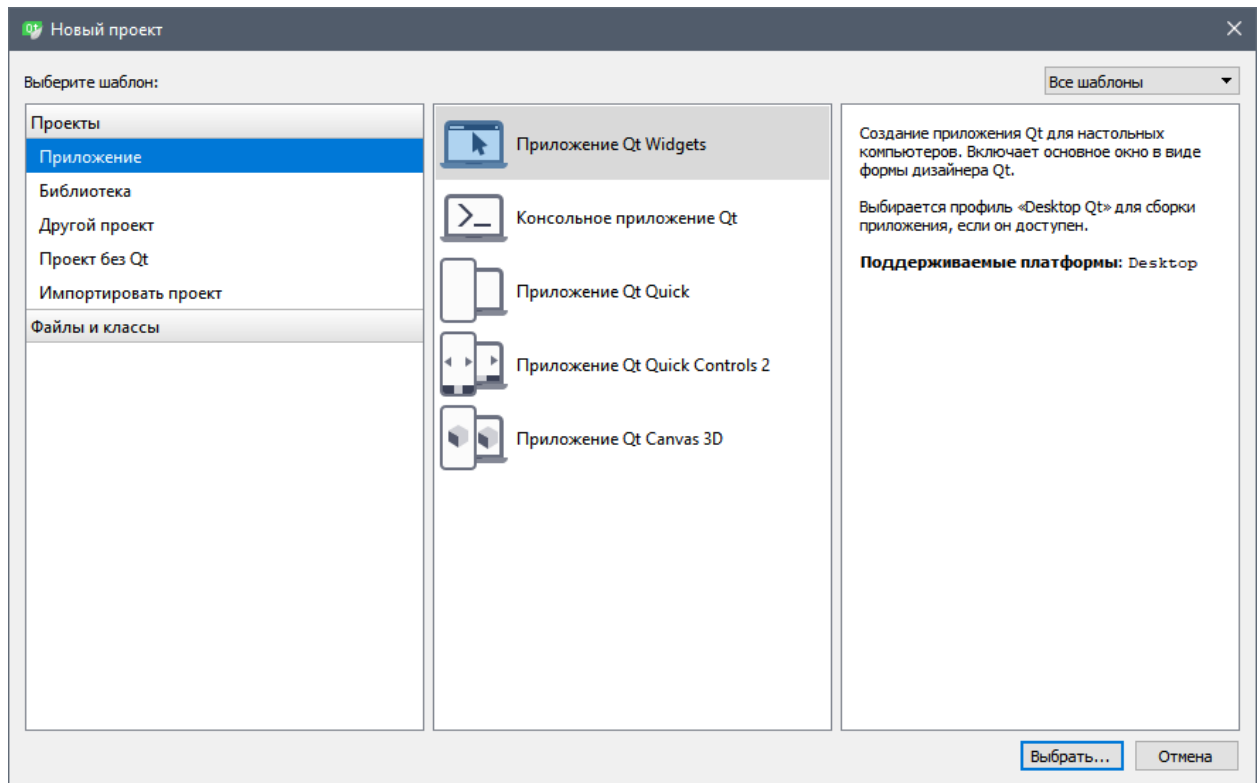


Рис. 1. Новый проект

Введите название проекта и выберите путь для размещения проекта.

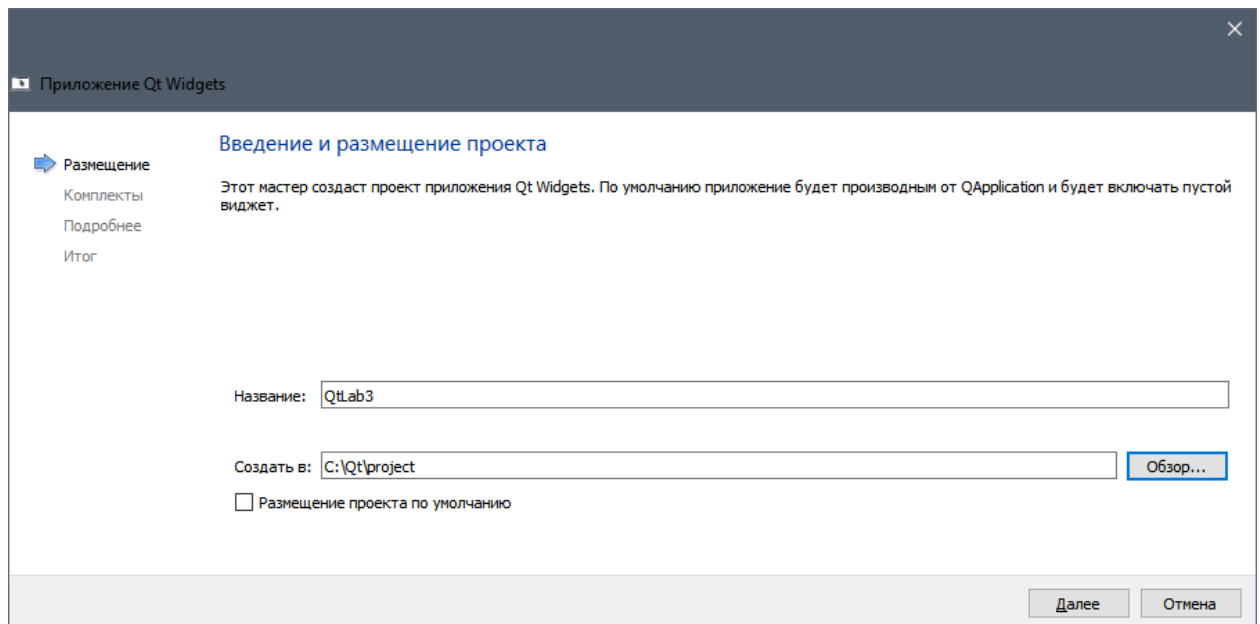


Рис. 2. Название проекта

Выберите компилятор (при наличии в системе нескольких компиляторов, для Qt Creator рекомендуется использовать MinGW).

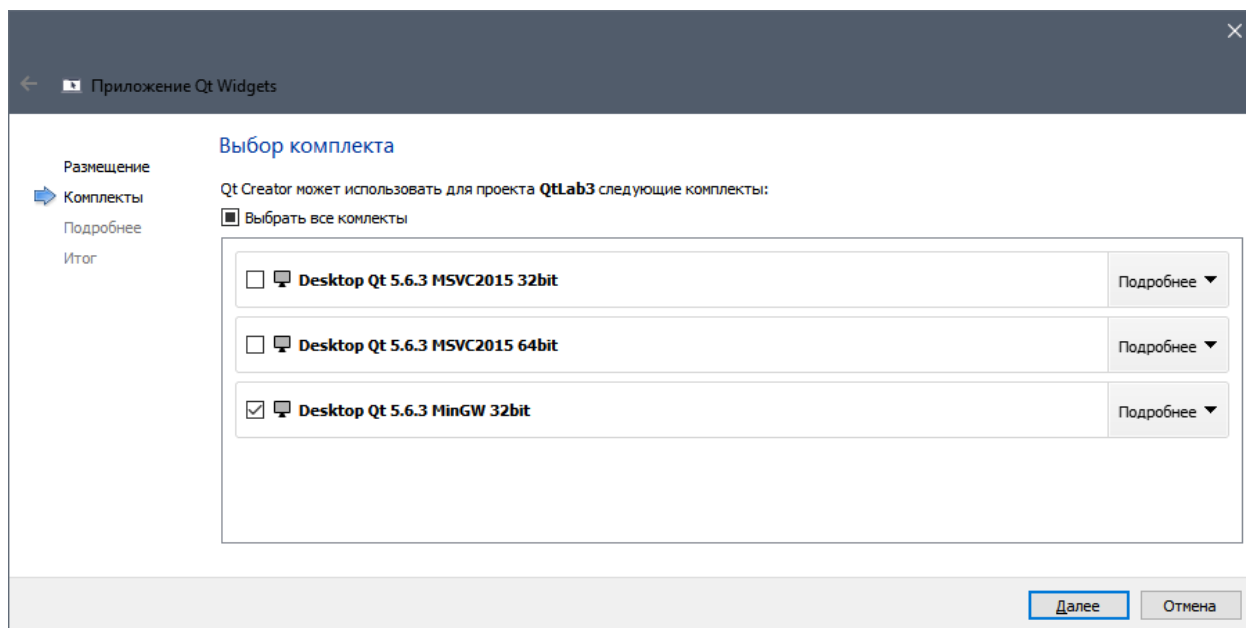
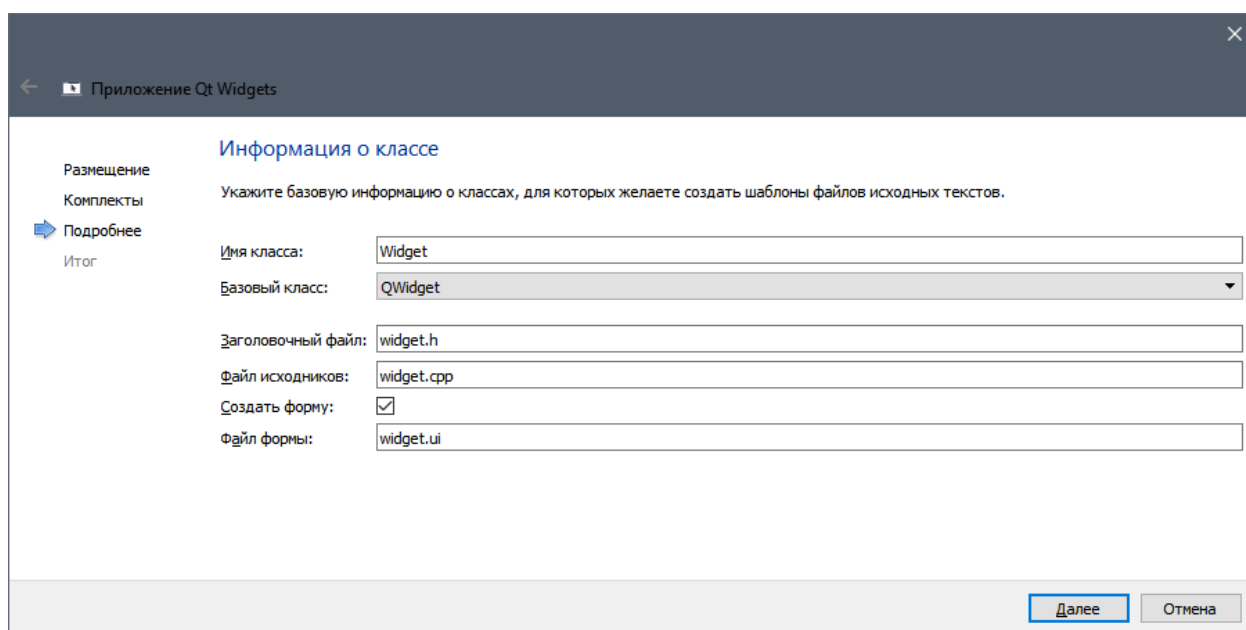
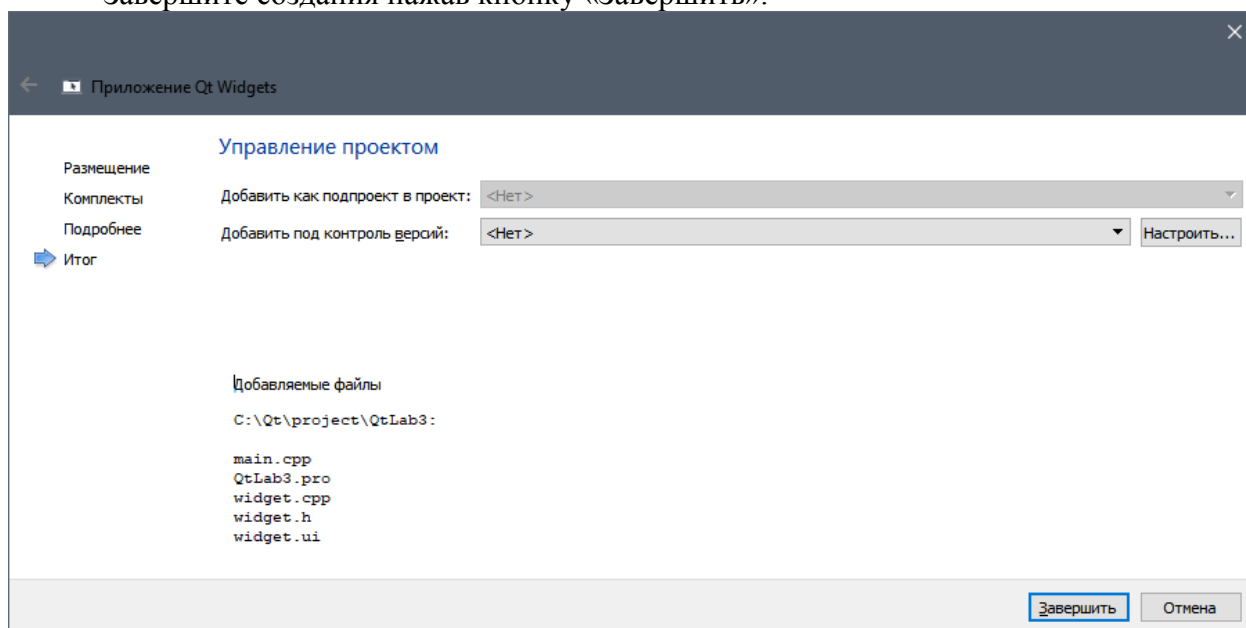


Рис. 3. Выбор компилятора

В качестве базового класса выберете QWidget. Файл заголовочный, исходников и формы оставьте названным по умолчанию (widget).



Завершите создания нажав кнопку «Завершить».



Создание интерфейса

После создания проекта, форма доступна для редактирования в двух режимах: через визуальный редактор внешнего вида (Дизайн) и путем редактирования кода формы (Редактор).

При редактировании внешнего вида в левой части экрана расположен набор экранных элементов. С помощью мыши они размещаются на форме. В правой верхней части экрана расположена иерархия объектов, которые размещены на форме. Доступ к свойствам размещенного элемента предоставляется через область «Свойства» в правой нижней части экрана. При выборе элемента он выделяется контуром (помечается синими квадратами) и доступен для перемещения. Для компоновки элементов на форме можно выбрать один из «макетов расположения» (Layouts). Так же доступна автоматическая компоновка в контекстном меню выбранного элемента, пункт «Компоновка».

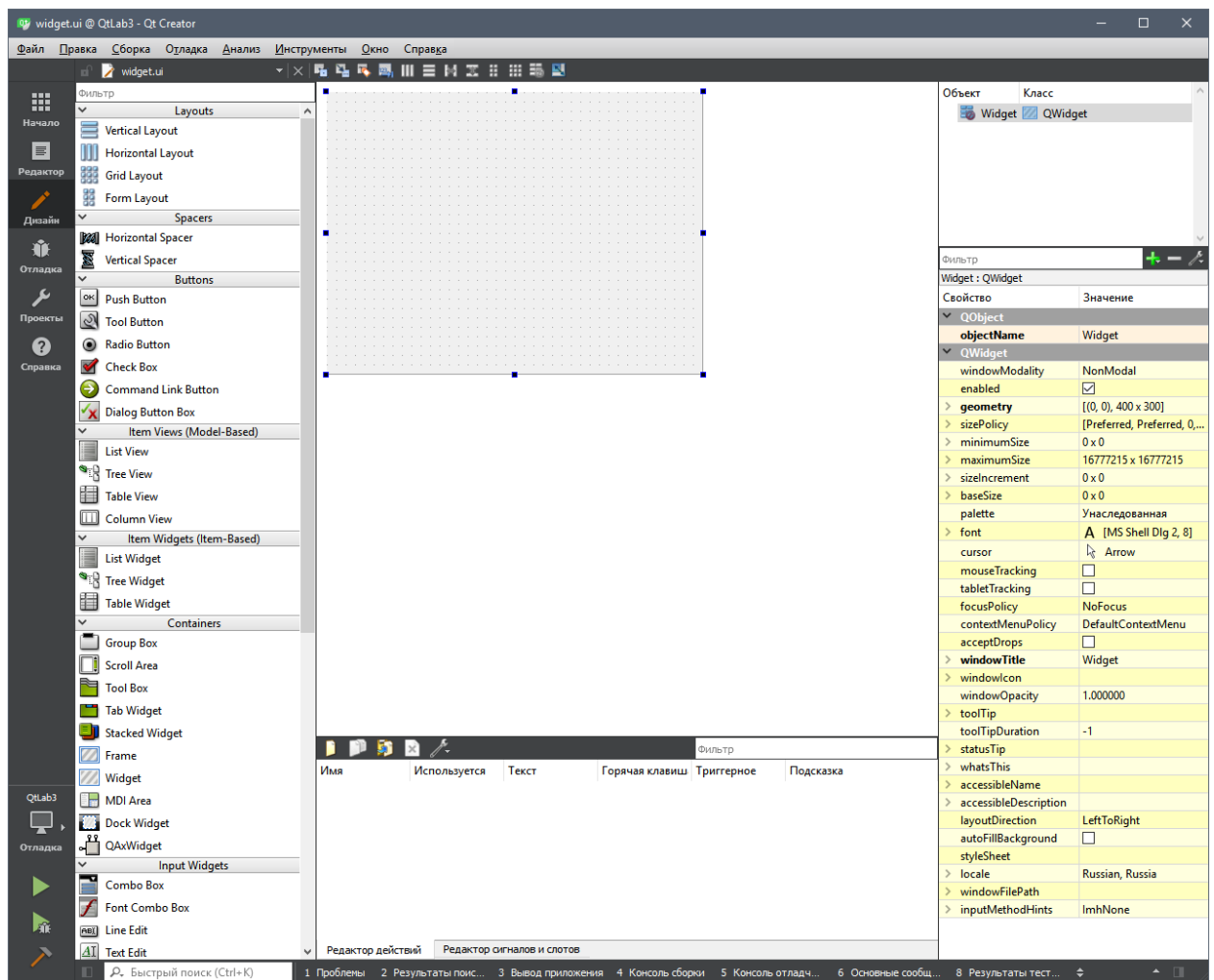
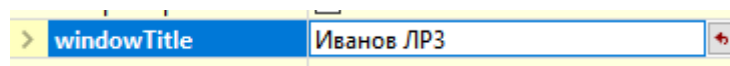


Рис. 4. Редактор форм в Qt Creator

В свойстве `windowTitle` Элемента «Widget» задайте заголовок формы, в соответствующий вашей фамилии.



Разместите на форме элементы управления. Для ровного размещения элементов на форме используйте Grid Layout.

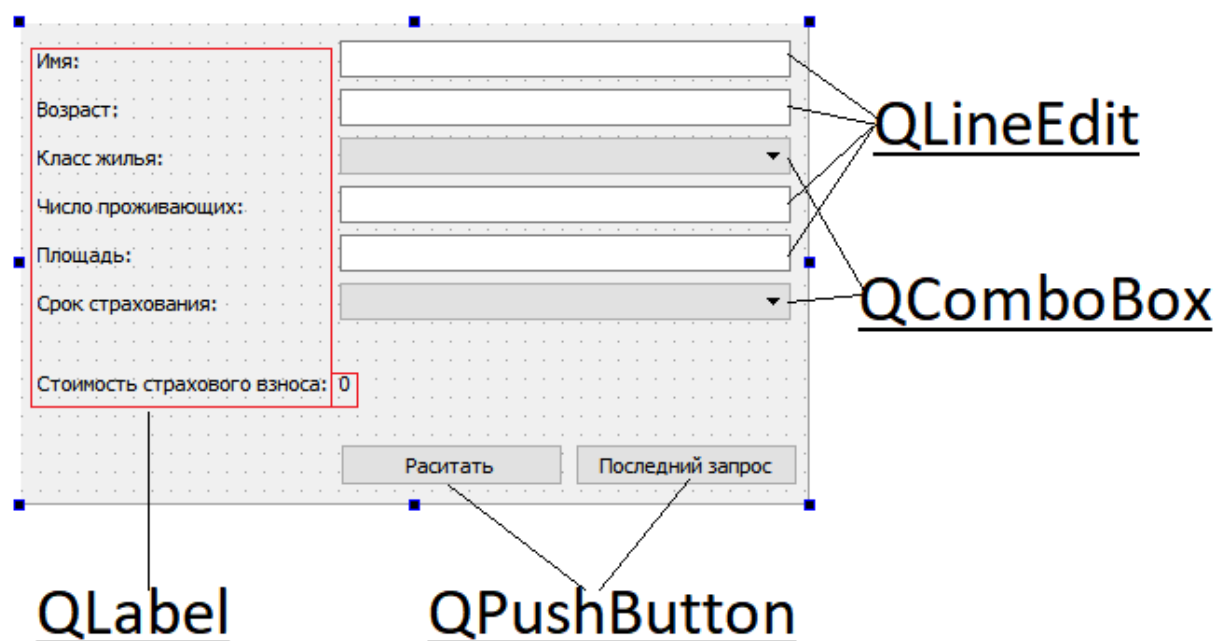


Рис. 5. Размещение элементов на форме

С помощью контекстного меню «Изменить objectName...» измените имена для каждого из объектов на форме, кроме подписей, расположенных справа.

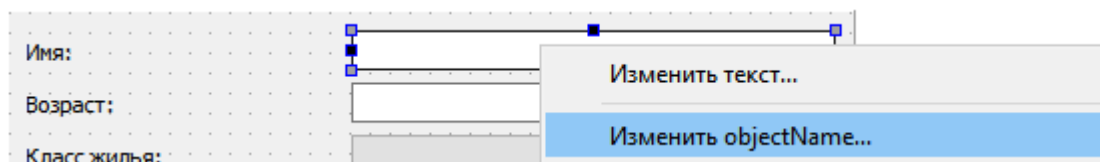


Рис. 6. Изменение имени элемента. Пример 1

Изменить имя элемента можно в свойствах.

| owner : QLineEdit | |
|-------------------|----------|
| Свойство | Значение |
| ▼ QObject | |
| objectName | owner |

Рис. 7. Изменение имени элемента. Пример 2

Имена элементов, которые в дальнейшем будут использованы.

Рис. 8. Имена элементов интерфейса

С помощью контекстного меню «Изменить текст» для элемента QLineEdit, QLabel, QPushButton или «Изменить элемент» для QComboBox установите указанные ниже значения по умолчанию.

Содержание для списка estateType (тип недвижимости) значения:

- Квартира эконом-класса
- Элитная квартира
- Таун-хаус
- Коттедж

Создайте для списка period (срок страхования) значения:

- 6 месяцев
- 1 год
- 18 месяцев

The image shows a web form with a grid background. It contains several input fields and buttons. The labels are on the left, and the values are on the right. The values are: 'Иванов' for 'Имя:', '25' for 'Возраст:', 'Квартира эконом-класса' for 'Класс жилья:', '56' for 'Число проживающих:', '170' for 'Площадь:', '6 месяцев' for 'Срок страхования:', and '0' for 'Стоимость страхового взноса:'. At the bottom right, there are two buttons: 'Расчитать' and 'Последний запрос'.

| | |
|------------------------------|------------------------|
| Имя: | Иванов |
| Возраст: | 25 |
| Класс жилья: | Квартира эконом-класса |
| Число проживающих: | 56 |
| Площадь: | 170 |
| Срок страхования: | 6 месяцев |
| Стоимость страхового взноса: | 0 |

Расчитать Последний запрос

Рис. 9. Пример формы, заполненной значениями по умолчанию

Бизнес-логика приложения

Предполагается создание учебной программы по расчету страховых взносов за жилье. Программа должна осуществлять расчет в зависимости от введенных пользователем данных, отображать результат расчета, а также иметь возможность возврата к предыдущим запросам пользователя.

Для этого понадобятся:

1. Функции расчета страховки;
2. Хранение данных о предыдущих запросах.

Создание класса параметров запроса

Создайте новый класс Estate и дополните его описанием типа. Для этого в обозревателе проектов выберите свой проект, в контекстном меню, выберите пункт «Добавить новый...». Выберите «Класс C++», в качестве базового класса QObject.

```
// Листинг файла estate.h
#ifndef ESTATE_H
#define ESTATE_H

#include <QObject>

class Estate : public QObject
{
    Q_OBJECT
public:
    enum EstateType {
        ECONOM,
        LUXURIOUS,
        TOWN_HOUSE,
        COTTAGE
    };
    explicit Estate(QObject *parent = nullptr);
    EstateType getType() const;

private:
    int age;
    int area;
    int residents;
    int months;
    EstateType type;
    QString owner;
};
#endif // ESTATE_H
```

Поле type имеет перечислимый тип, соответствующий типам недвижимости в предметной области. Обратите внимание, что значение в перечислимом типе недвижимости и индексом записи в выпадающем списке интерфейсной формы.

Задание. Дополните класс Estate конструктором, заполняющим все его поля и функциями чтения полей.

Сохранение информации о предыдущих запросах

Для хранения информации о предыдущих запросах, предлагается использовать коллекцию элементов переменной длины – QList. После создания коллекции, для добавления элементов, обращения по индексу и изъятия применяются методы: append, takeAt, removeAt. Число элементов в коллекции определяется методом size. Создадим класс States.

```
// Листинг файла states.h
#include <QObject>
#include <estate.h>

class States : public QObject
{
    Q_OBJECT
public:
    explicit States(QObject *parent = nullptr);
    ~States();

    void undo();
    bool hasStates();
    Estate *getActualData();
    void add(Estate *value);
private:
    QList<Estate *> array;
    Estate *actualData;
};

// Листинг файла states.cpp
#include "states.h"

States::States(QObject *parent) : QObject(parent)
{
    actualData = nullptr;
}

States::~States()
{
    // delete: actualData
    if(actualData){
        delete actualData;
        actualData = nullptr;
    }
    // delete and cleare: arra
    qDeleteAll(array);
    array.clear();
}
```

QList<Estate *> array – объявление коллекции, которая может хранить элементы класса Estate. Тип хранимых данных в скобках позволяет поручить компилятору следить за типом добавляемых данных.

hasStates – возвращает истину (true), если в коллекции есть элементы.

getActualData – возвращает последний элемент коллекции (с наибольшим индексом).

add – добавление элемента в коллекцию.

Поле actualData будет содержать последний запрос при передаче его в другую часть приложения. Поэтому метод undo должен (Метод undo в последующем подвергнется доработке):

1. Устанавливать actualData в null, если коллекция пуста, иначе

2. Инициализировать `actualData` значениями, соответствующими данным последнего элемента коллекции. Удалять из коллекции последний элемент.

Задание. Реализуйте методы `undo`, `hasStates`, `getActualData`, `add`.

Создание фасада для подсчета суммы страховых взносов

Предполагается, что расчет суммы взносов может быть осуществлен на основе содержимого класса `Estate`.

Расчет взносов может производиться различным образом для разных типов недвижимости, с учетом или не учетом площади, стоимостной группы, числа проживающих и прочего. Таким образом, может существовать группа функций расчета.

Предположим также, что процедура расчета взносов может меняться чаще, чем остальные части информационной системы, реагируя на рыночные изменения и политику предприятия. В такой ситуации полезным может оказаться использование шаблона проектирования Фасад (Facade).

Шаблон Facade транслирует запросы клиентов тем подсистемам, которые могут их обслужить. В большинстве случаев один запрос делегируется сразу нескольким подсистемам. Поскольку клиент взаимодействует лишь с классом Facade, это позволяет менять внутреннюю логику системы, не затрагивая принципов работы клиента с классом Facade.

Применение шаблона Facade способствует снижению взаимосвязи клиента с подсистемами. Кроме того, этот шаблон с успехом можно применять и для снижения взаимосвязей между самими подсистемами. Каждая подсистема может иметь собственный экземпляр класса Facade, через который с ней взаимодействуют классы из других частей системы.

В рамках лабораторной работы предлагается использовать шаблон Facade для применения той или иной схемы расчета.

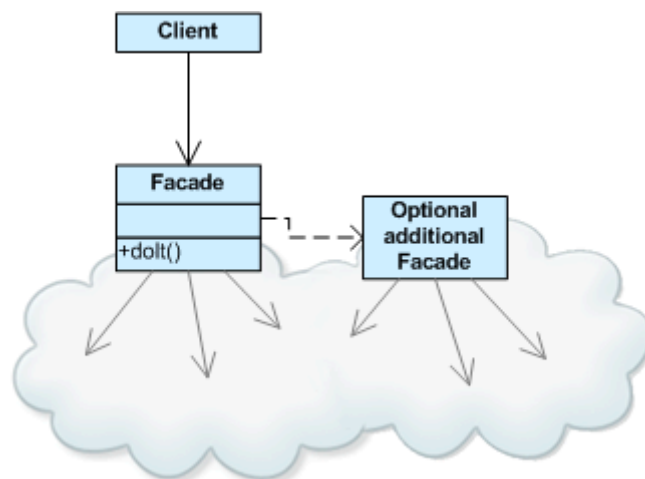


Рис. 10. Диаграмма классов паттерна Facade

Создадим класс `CalculationFacade` следующего вида:

```
// Листинг файла calculationfacade.h
#ifndef CALCULATIONFACADE_H
#define CALCULATIONFACADE_H
```

```

#include <QObject>
#include <estate.h>
#include <apartmentcalc.h>
#include <luxuriousapartmentcalc.h>
#include <townhousecalc.h>
#include <cottagecalc.h>

class CalculationFacade : public QObject
{
    Q_OBJECT
public:
    explicit CalculationFacade(QObject *parent = nullptr);

    static int getCost(Estate *value);
};

#endif // CALCULATIONFACADE_H

```

// Листинг файла calculationfacade.cpp

```

#include "calculationfacade.h"

CalculationFacade::CalculationFacade(QObject *parent) : QObject(parent)
{
}

int CalculationFacade::getCost(Estate *value)
{
    int cost;

    switch (value->getType()) {
    case Estate::EstateType::ECONOM:
        cost = ApartmentCalc::getCost(value);
        break;

    case Estate::EstateType::LUXURIOUS:
        cost = LuxuriousApartmentCalc::getCost(value);
        break;

    case Estate::EstateType::TOWN_HOUSE:
        cost = TownhouseCalc::getCost(value);
        break;

    case Estate::EstateType::COTTAGE:
        cost = CottageCalc::getCost(value);
        break;

    default:
        cost = -1;
        break;
    }

    return cost;
}

```

Задание. Реализуйте собственные функции расчета стоимости в отдельном классе каждая и организуйте их вызов из класса-фасада. Класс-фасад может быть изменен, но

передача данных для расчета должна производиться в виде объекта Estate, как показано в примере выше.

Лабораторная работа №9

«Поведенческие шаблоны проектирования»

Цель работы

Изучить принципы построения приложений с графическим интерфейсом, используя библиотеку Qt, применив на практике знания базовых синтаксических конструкций языка C++ и объектно-ориентированного программирования.

Закрепить знания по теме: Структурные, поведенческие и порождающие шаблоны проектирования.

Требования к лабораторной работе:

Выполнить все требования к лабораторной работе, описанные в ее формулировке, подготовить объяснение структуры программы и принципов ее функционирования, продемонстрировать рабочую программу. Подготовить отчет в соответствии с шаблоном.

Описание работы

Шаблон проектирования, паттерн (англ. design pattern) — это многократно применяемая архитектурная конструкция, предоставляющая решение общей проблемы проектирования в рамках конкретного контекста и описывающая значимость этого решения. Паттерн не является законченным образцом проекта, который может быть прямо преобразован в код.

Это описание или образец для того, как решить задачу таким образом, чтобы это можно было использовать в различных ситуациях. Объектно-ориентированные шаблоны зачастую показывают отношения и взаимодействия между классами или объектами, без определения того, какие конечные классы или объекты приложения будут использоваться.

Основными типами шаблонов проектирования являются:

- порождающие шаблоны;
- структурные шаблоны;
- поведенческие шаблоны.

Приложение, создаваемое в лабораторной работе, предназначено для знакомства с шаблоном проектирования Наблюдатель. Программа имитирует расчет страховых взносов. Пользователь вводит на форме данные страховщика, запускает расчет страховых взносов, а также имеет возможность вернуться к прежним запросам. Лабораторные работы № 8 – 10 предполагают последовательное выполнение и получение в конце рабочего экземпляра программы.

Информация о паттернах проектирования

1. <http://cpp-reference.ru/patterns/>
2. <http://citforum.ru/SE/project/pattern/#anno>

Организация взаимодействия формы и класса данных. Паттерн наблюдатель.

Главная форма приложения должна выполнять функции приема пользовательских данных, запроса расчета у класса CalculationFacade, а также взаимодействия с классом хранения данных: добавления туда новых запросов и изъятия старых (функция Undo/Последний запрос). Отметим, что при откате возврат вперед не предполагается, то есть коллекция прежних запросов очищается при каждом возврате.

Несмотря на то, что в учебном приложении взаимодействие компонент предельно упрощено, в реальной системе возврат данных может происходить через сложную процедуру запроса на удаленную систему. В таком случае используют асинхронные механизмы связи, при котором запрос и возврат результатов могут быть непоследовательными.

В приложении используем реализацию подобного механизма через шаблон Наблюдатель (Observer), являющийся поведенческим шаблоном.

Шаблон Observer лучше всего применять в тех случаях, когда система обладает следующими свойствами.

- Существует, как минимум, один объект, рассылающий сообщения.
- Имеется не менее одного получателя сообщений, причем их количество и состав может изменяться во время работы приложения, а также различаться у разных экземпляров приложения.

Данный шаблон часто применяют в ситуациях, в которых отправитель сообщений не должен или не хочет знать, что делают получатели с предоставленной им информацией. Его задача — лишь разослать информацию всем, кому она нужна.

Некоторые отправители сообщений, называемые также генераторами сообщений (message producers), работают в соответствии с простой моделью одноранговых коммуникаций, создавая сообщения, которые предназначены строго определенным получателям сообщений (message receiver). В таких случаях организовать обработку событий очень просто. Однако бывают генераторы сообщений, чье поведение далеко не так просто. Отправка сообщения может повлечь за собой переменное количество последовательных ответных реакций, в которых задействуются как генератор, так и один или более получателей.

Шаблон Observer можно представить, как решение, основанное на доставке информации сервером (server-push). Сервер (в данном случае — наблюдаемый объект, или генератор событий) контактирует с заинтересованным наблюдателем в тех случаях, когда возникает то или иное событие.

Шаблон Observer может оказаться полезным в самых разных приложениях. Так как извещения рассылаются только тем элементам, которые идентифицировали себя в качестве заинтересованных получателей, последние могут реагировать на полученную информацию так, как они считают нужным. Данный подход хорошо применим к любой модели, в которой, во-первых, изменения одного компонента могут приводить к изменению других компонентов, а во-вторых, поведение объектов может настраиваться во время работы приложения.

С точки зрения бизнес-модели шаблон Observer полезен в тех случаях, когда в модели реализуются сложные операции изменения, удаления или обновления информации. Гибкая природа шаблона позволяет применять его для рассылки информации как некоторым, так и всем элементам модели.

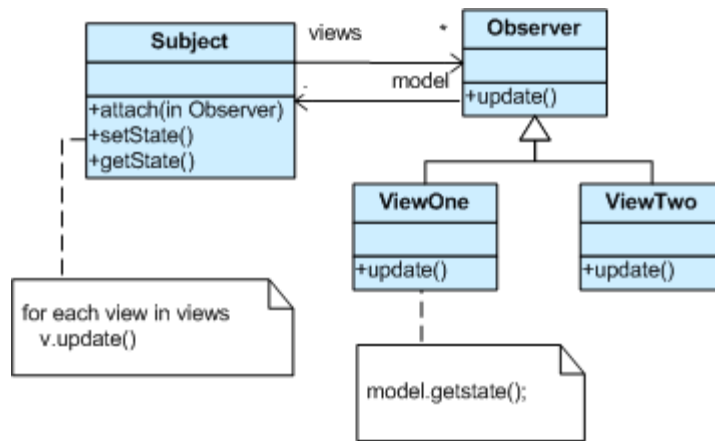


Рис. 11. Диаграмма классов шаблона Observer

В нашем классе States должен реализовать метод-сигнал (signals) void notifyObservers(). Соответственно, функция undo класса States после заполнения actualData должна вызывать эмиттер (emit notifyObservers()) сигнал: notifyObservers – уведомление слушателей о произошедшем событии изменения данных.

Класс главной формы должен реализовывать метод-слот (slots) void update(), вызываемый наблюдаемым классом при возникновении события (notifyObservers).

Предлагается использовать следующую структуру как базовую для класса формы:

// Листинг файла widget.h

```

#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <states.h>
#include <estate.h>
#include <calculationfacade.h>
namespace Ui {
class Widget;
}
class Widget : public QWidget
{
    Q_OBJECT
public:
    explicit Widget(QWidget *parent = 0);
    ~Widget();

public slots:
    void update();
private slots:
    void btnCalcPressed();
    void btnUndoPressed();
private:
    Estate *processForm();
    fillForm(Estate *value);
    showCost(Estate *value);
private:
    Ui::Widget *ui;

    States info;
};
  
```

```
#endif // WIDGET_H
```

// Листинг файла widget.cpp

```
#include "widget.h"
#include "ui_widget.h"
```

```
Widget::Widget(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::Widget),
    info(this)
{
    ui->setupUi(this);
    ui->btnUndo->setEnabled(false);

    // регистрация слушателя
    connect(&info, SIGNAL(notifyObservers()), this, SLOT(update()));
    connect(ui->btnCalc, SIGNAL(pressed()), this, SLOT(btnCalcPressed()));
    connect(ui->btnUndo, SIGNAL(pressed()), this, SLOT(btnUndoPressed()));
}

Widget::~Widget()
{
    delete ui;
}

// public slots
void Widget::update()
{
    auto value = info.getActualData();
    if(value != nullptr){
        fillForm(value);
    }
    // update btnUndo state
    ui->btnUndo->setEnabled(info.hasStates());

    // setting value to NULL
    value = nullptr;
}

// private slots
void Widget::btnCalcPressed()
{
    auto value = processForm();
    showCost(value);
    info.add(value);
    ui->btnUndo->setEnabled(true);

    // setting value to NULL
    value = nullptr;
}

void Widget::btnUndoPressed()
{
    info.undo();
}

// private
Estate *Widget::processForm()
{
    return new Estate();
}

Widget::fillForm(Estate *value)
{
}

Widget::showCost(Estate *value)
{
}
```

```
}
```

Обратите внимание, что при создании формы она регистрируется в качестве слушателя в объекте класса States.

```
connect(&info, SIGNAL(notifyObservers()), this, SLOT(update()));
```

- update – вызывается при поступлении сигнала, после выполняется взаимодействие с States и заполнение данных на форме.
- processForm – обрабатывает данные формы, создает объект класса Estate;
- fillForm – отображает данные объекта класса Estate на форме.
- showCost – отображает стоимость в соответствующей метке формы, получая стоимость от класса CalculationFacade.

Задание. Реализуйте функции processForm, fillForm, showCost. Убедитесь в том, что приложение реализует функциональность, заявленную во введении.

Для работы со значением поля ввода используйте метод text/setText:

```
QString str = ui->age->text();  
ui->age->setText(str);
```

Перевод числа в строку и строки в число:

```
int i = ui->age->text().toInt();  
ui->age->setText(QString::number(i));
```

Лабораторная работа №10

«Порождающие шаблоны проектирования»

Цель работы

Изучить принципы построения приложений с графическим интерфейсом, используя библиотеку Qt, применив на практике знания базовых синтаксических конструкций языка C++ и объектно-ориентированного программирования.

Закрепить знания по теме: Структурные, поведенческие и порождающие шаблоны проектирования.

Требования к лабораторной работе:

Выполнить все требования к лабораторной работе, описанные в ее формулировке, подготовить объяснение структуры программы и принципов ее функционирования, продемонстрировать рабочую программу. Подготовить отчет в соответствии с шаблоном.

Описание работы

Шаблон проектирования, паттерн (англ. design pattern) — это многократно применяемая архитектурная конструкция, предоставляющая решение общей проблемы проектирования в рамках конкретного контекста и описывающая значимость этого решения. Паттерн не является законченным образцом проекта, который может быть прямо преобразован в код.

Это описание или образец для того, как решить задачу таким образом, чтобы это можно было использовать в различных ситуациях. Объектно-ориентированные шаблоны зачастую показывают отношения и взаимодействия между классами или объектами, без определения того, какие конечные классы или объекты приложения будут использоваться.

Основными типами шаблонов проектирования являются:

- порождающие шаблоны;
- структурные шаблоны;
- поведенческие шаблоны.

Приложение, создаваемое в лабораторной работе, предназначено для знакомства с шаблоном проектирования Фабричный метод. Программа имитирует расчет страховых взносов. Пользователь вводит на форме данные страховщика, запускает расчет страховых взносов, а также имеет возможность вернуться к прежним запросам.

Информация о паттернах проектирования

1. <http://cpp-reference.ru/patterns/>
2. <http://citforum.ru/SE/project/pattern/#anno>

Паттерн Factory Method (Фабричный метод)

В системе часто требуется создавать объекты самых разных типов. Паттерн Factory Method (фабричный метод) может быть полезным в решении следующих задач:

- Система должна оставаться расширяемой путем добавления объектов новых типов. Непосредственное использование выражения `new` является нежелательным, так как в этом случае код создания объектов с указанием конкретных типов может получиться разбросанным по всему приложению. Тогда такие операции как добавление в систему объектов новых типов или замена объектов одного типа на другой будут затруднительными (подробнее в разделе Порождающие паттерны). Паттерн Factory Method позволяет системе оставаться независимой как от самого процесса порождения объектов, так и от их типов.
- Заранее известно, когда нужно создавать объект, но неизвестен его тип.

Для того, чтобы система оставалась независимой от различных типов объектов, паттерн Factory Method использует механизм полиморфизма - классы всех конечных типов наследуют от одного абстрактного базового класса, предназначенного для полиморфного использования. В этом базовом классе определяется единый интерфейс, через который пользователь будет оперировать объектами конечных типов.

Для обеспечения относительно простого добавления в систему новых типов паттерн Factory Method локализует создание объектов конкретных типов в специальном классе-фабрике. Методы этого класса, посредством которых создаются объекты конкретных классов, называются фабричными.

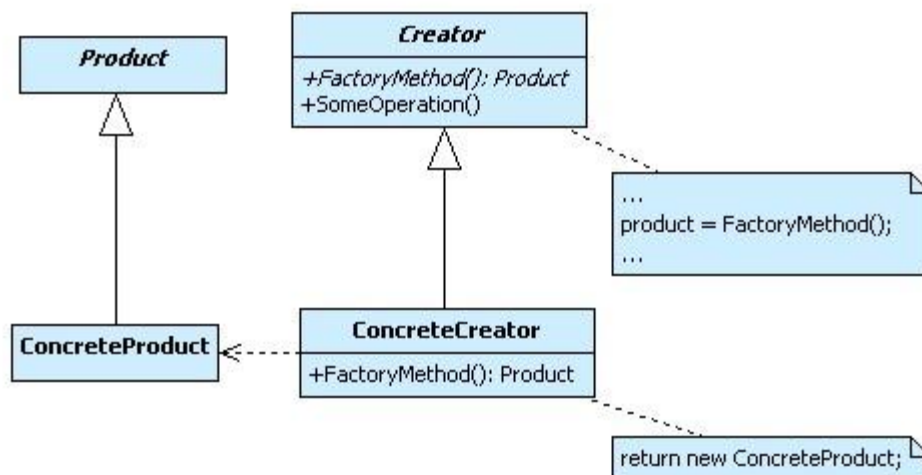


Рис. 12. Диаграмма классов шаблона Factory Method

Задание. Необходимо изменить вызов статических функций `getCost` в `CalculationFacade`, на реализацию Factory Method. Для этого необходимо использовать базовые классы `bstractCalc` и `CalcFactory` реализующие интерфейсы паттерна Factory Method. В `CalculationFacade`,

вместо статических функций будет вызываться фабрика необходимого для расчётов класса, создаваться объект класса, а у него вызов метода `getCost`.

Приложение 1

Учебная программа «Библиотека фигур»

```
//Файл point.h
#ifndef POINT_H
#define POINT_H

#include <stdint>

class Point //Точка на экране
{
public:
    Point() : _x(0), _y(0) { };
    Point(std::uint32_t x, std::uint32_t y) : _x(x), _y(y) {}

    std::uint32_t getX() const { return _x; }
    std::uint32_t getY() const { return _y; }
    void setX(std::uint32_t val) { _x = val; }
    void setY(std::uint32_t val) { _y = val; }

protected:
    std::uint32_t _x, _y;
};

#endif // POINT_H

//Файл screen.h
#ifndef SCREEN_H
#define SCREEN_H

#include "point.h"

#include <vector>
#include <iostream>

class Screen
{
public:
    enum class Pixel : char
    {
        WHITE = ' ',
    }
};
```



```

        BLACK = '*';
    };

Screen(std::uint32_t xSize, std::uint32_t ySize)
{
    _xSize = xSize;
    _ySize = ySize;
    _screen.reserve(ySize);
    for (size_t i = 0; i < ySize; i++)
        _screen.emplace_back(std::vector<Pixel>(xSize, Pixel::WHITE));
}

void putLine(const Point& a, const Point& b)
{
    putLine(a.getX(), a.getY(), b.getX(), b.getY());
}

void putLine(std::uint32_t x0, std::uint32_t y0, std::uint32_t x1, std::uint32_t
y1)
{
    std::int32_t dx = 1;
    std::int32_t a = x1 - x0;
    if (a < 0) { dx = -1; a = -a; }
    std::int32_t dy = 1;
    std::int32_t b = y1 - y0;
    if (b < 0) { dy = -1; b = -b; }
    std::int32_t two_a = 2 * a;
    std::int32_t two_b = 2 * b;
    std::int32_t xcrit = -b + two_a;
    std::int32_t eps = 0;

    while (true)
    {
        putPoint(x0, y0);
        if (x0 == x1 && y0 == y1) break;
        if (eps <= xcrit)
        {
            x0 += dx;
            eps += two_b;
        }
    }
}

```

```

        if (eps >= a || a < b)
        {
            y0 += dy;
            eps -= two_a;
        }
    }

}

void putPoint(const Point& p)
{
    _screen[p.getX()][p.getY()] = Pixel::BLACK;
}

void putPoint(const std::uint32_t x, const std::uint32_t y)
{
    _screen[x][y] = Pixel::BLACK;
}

void clear()
{
    for (size_t i = 0; i < _ySize; ++i)
        for (size_t j = 0; j < _xSize; ++j)
            _screen[i][j] = Pixel::WHITE;
}

void draw() const
{
    for (size_t i = 0; i < _ySize; ++i)
    {
        for (size_t j = 0; j < _xSize; ++j)
            std::cout << (char)_screen[i][j];
        std::cout << std::endl;
    }
}

private:
    std::uint32_t _xSize;
    std::uint32_t _ySize;
    std::vector<std::vector<Pixel> > _screen;
};

```

```

#endif // SCREEN_H

// Файл shape.h
#ifndef SHAPE_H
#define SHAPE_H

#include "screen.h"

class Shape
{
public:
    virtual void draw(Screen* screen) const = 0;

    virtual void move(Point p) = 0;

    virtual Point getLeftTop() const = 0;

    virtual Point getRightTop() const = 0;

    virtual Point getLeftBottom() const = 0;

    virtual Point getRightBottom() const = 0;

    virtual ~Shape() {}
};

#endif // SHAPE_H

// Файл rotatable.h
#ifndef ROTATABLE_H
#define ROTATABLE_H

class Rotatable { //Фигуры, пригодные к повороту
public:
    virtual void rotateLeft() = 0;    //Повернуть влево
    virtual void rotateRight() = 0;   //Повернуть вправо
};

#endif // ROTATABLE_H

```

```

// Файл reflectable.h
#ifndef REFLECTABLE_H
#define REFLECTABLE_H
class reflectable { // Фигуры, пригодные к зеркальному отражению

public:
    virtual void flipHorisontally() = 0;    // Отразить горизонтально
    virtual void flipVertically() = 0;      // Отразить вертикально
};

#endif // REFLECTABLE_H

// Файл line.h
#ifndef LINE_H
#define LINE_H

#include "shape.h"
#include "screen.h"

#include <algorithm>

class Line : public Shape
{
public:
    Line(const Point& a, const Point& b)
    {
        _a = a;
        _b = b;
    }

    Line(unsigned int x0, unsigned int y0, unsigned int x1, unsigned int y1)
    {
        _a = Point(x0, y0);
        _b = Point(x1, y1);
    }

    Line(const Line& source)
    {
        _a = source._a;
        _b = source._b;
    }

```

```

    }

    void draw(Screen* screen) const
    {
        screen->putLine(_a.getY(), _a.getX(), _b.getY(), _b.getX());
    }

    void move(Point p)
    {
        _a.setX(_a.getX() + p.getX()); _a.setY(_a.getY() + p.getY());
        _b.setX(_b.getX() + p.getX()); _b.setY(_b.getY() + p.getY());
    }

    Point getLeftTop() const
    {
        return Point(std::min(_a.getX(), _b.getX()), std::min(_a.getY(),
_b.getY()));
    }

    Point getRightTop() const
    {
        return Point(std::max(_a.getX(), _b.getX()), std::min(_a.getY(),
_b.getY()));
    }

    Point getLeftBottom() const
    {
        return Point(std::min(_a.getX(), _b.getX()), std::max(_a.getY(),
_b.getY()));
    }

    Point getRightBottom() const
    {
        return Point(std::max(_a.getX(), _b.getX()), std::max(_a.getY(),
_b.getY()));
    }

    Point getFirstPoint() const
    {
        return _a;
    }

```

```

        Point getSecondPoint() const
        {
            return _b;
        }

protected:
    Point _a;
    Point _b;
};

#endif // LINE_H

// Файл cross.h
#ifndef CROSS_H
#define CROSS_H

#include "shape.h"
#include "line.h"

//Крест
class Cross : public virtual Shape
{
public:
    Cross(const Point& left, const Point& top)
    {
        Point p1, p2;
        Point p3, p4;

        p1 = left;
        p2 = Point(top.getX() + top.getX() - left.getX(), left.getY());
        p3 = top;
        p4 = Point(top.getX(), left.getY() - top.getY() + left.getY());

        _first = new Line(p1, p2);
        _second = new Line(p3, p4);
    }

    virtual void draw(Screen* screen) const
    {
        _first->draw(screen);
        _second->draw(screen);
    }
};

```

```

    }

    virtual void move(Point p)
    {
        _first->move(p);
        _second->move(p);
    }

    Point getLeftTop() const
    {
        return _first->getLeftTop();
    }

    Point getRightTop() const
    {
        return _second->getRightTop();
    }

    Point getLeftBottom() const
    {
        return _second->getLeftBottom();
    }

    Point getRightBottom() const
    {
        return _first->getRightBottom();
    }

    virtual ~Cross()
    {
        delete _first;
        delete _second;
    }
protected:
    Line* _first;
    Line* _second;
};

#endif // CROSS_H

//Файл square.h

```

```

#ifndef SQUARE_H
#define SQUARE_H

#include "line.h"

class Square : public virtual Shape
{
public:
    Square(const Point& leftTop, const Point& rightBottom)
    {
        _left = new Line(leftTop, Point(leftTop.getX(), rightBottom.getY()));
        _top = new Line(leftTop, Point(rightBottom.getX(), leftTop.getY()));
        _right = new Line(Point(rightBottom.getX(), leftTop.getY()),
rightBottom);
        _bottom = new Line(Point(leftTop.getX(), rightBottom.getY()),
rightBottom);
    }

    virtual void draw(Screen* screen) const
    {
        _left->draw(screen);
        _top->draw(screen);
        _right->draw(screen);
        _bottom->draw(screen);
    }

    virtual void move(Point p)
    {
        _left->move(p);
        _top->move(p);
        _right->move(p);
        _bottom->move(p);
    }

    Point getLeftTop() const
    {
        return _left->getLeftTop();
    }

    Point getRightTop() const
    {

```



```

        return _right->getRightTop();
    }

    Point getLeftBottom() const
    {
        return _left->getLeftBottom();
    }

    Point getRightBottom() const
    {
        return _right->getRightBottom();
    }

    virtual ~Square()
    {
        delete _left;
        delete _top;
        delete _right;
        delete _bottom;
    }

protected:
    Line *_left, *_top, *_right, *_bottom;
};

#endif //SQUARE_H

// Файл main.cpp
#include "screen.h"
#include "line.h"
#include "square.h"

#include <vector>
#include <memory>

int main()
{
    auto screen = std::make_unique<Screen>(50, 35);

    std::vector<std::shared_ptr<Shape>> shapes;

```

```

shapes.emplace_back(std::make_shared<Square>(Point(15, 3), Point(32, 12))); //
Шляпа

Point p1 = shapes[shapes.size() - 1]->getLeftBottom();
p1.setX(p1.getX() - 2);
p1.setY(p1.getY() + 1);
Point p2 = shapes[shapes.size() - 1]->getRightBottom();
p2.setX(p2.getX() + 2);
p2.setY(p2.getY() + 1);
shapes.emplace_back(std::make_shared<Line>(p1, p2)); // Линия под шляпой

p1 = shapes[shapes.size() - 1]->getLeftBottom();
p1.setX(p1.getX() + 1);
p1.setY(p1.getY() + 1);
p2 = shapes[shapes.size() - 1]->getRightBottom();
p2.setX(p2.getX() - 1); p2.setY(p2.getY() + 10);
shapes.emplace_back(std::make_shared<Square>(p1, p2)); // Голова

Point eyeLeft = shapes[shapes.size() - 1]->getLeftTop();
eyeLeft.setX(eyeLeft.getX() + 2);
eyeLeft.setY(eyeLeft.getY() + 2);
Point eyeRight = Point(eyeLeft.getX() + 2, eyeLeft.getY());
shapes.emplace_back(std::make_shared<Line>(eyeLeft, eyeRight)); // Левый глаз

eyeRight = shapes[shapes.size() - 2]->getRightTop();
eyeRight.setX(eyeRight.getX() - 2);
eyeRight.setY(eyeRight.getY() + 2);
eyeLeft = Point(eyeRight.getX() - 2, eyeRight.getY());
shapes.emplace_back(std::make_shared<Line>(eyeLeft, eyeRight)); // Правый глаз

std::shared_ptr<Shape> leftEye = shapes[shapes.size() - 2];
std::shared_ptr<Shape> rightEye = shapes[shapes.size() - 1];
Point nose = Point(leftEye->getRightTop().getX() +
    (rightEye->getLeftTop().getX() - leftEye->getRightTop().getX()) / 2,
    leftEye->getRightTop().getY() + 2);
shapes.emplace_back(std::make_shared<Line>(nose, nose)); // Нос

std::shared_ptr<Shape> head = shapes[shapes.size() - 4];
p1 = head->getLeftBottom(); p1.setX(p1.getX() + 2); p1.setY(p1.getY() - 2);
p2 = head->getRightBottom(); p2.setX(p2.getX() - 2); p2.setY(p2.getY() - 2);
shapes.emplace_back(std::make_shared<Line>(p1, p2)); // Рот

```

```

    auto costume = std::make_shared<Line>(Point(p1.getX(), p1.getY() + 10),
        Point(p2.getX(), p2.getY() + 10));
    shapes.emplace_back(costume); //Линия костюма

1);
    p1 = Point(costume->getLeftBottom().getX(), costume->getLeftBottom().getY() +
1);
    auto leftDot = std::make_shared<Line>(p1, p1);
    shapes.emplace_back(leftDot); //Левая точка

    p1 = Point(costume->getRightBottom().getX(), costume->getRightBottom().getY() +
1);
    auto rightDot = std::make_shared<Line>(p1, p1);
    shapes.emplace_back(rightDot); //Правая точка

    for (auto shape : shapes)
        shape->draw(screen.get());
    screen->draw();

    std::cin.get();
    return 0;
}

```