**1. Постановка задачи Описание постановки задачи.**

1.1

Запрограммируйте шаблонный класс, реализующий стек. Класс должен поддерживать следующие операции: помещение объекта в стек, извлечение объекта из стека, получение размерности стека. В случае попытки вызова операции извлечение объекта из стека при условии, что стек пуст, должно генерироваться исключение класса EStackEmpty. Данный класс должен содержать публичный метод char\* what(), возвращающий диагностическое сообщение.  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1.2

Реализовать класс PersonKeeper с методами readPersons и writePersons. Метод readPersons должен считывать информацию о людях из входного потока (файла), создавать на основе этой информации объекты класса Person, и помещать их в стек. Формат входного файла должен быть такой:

Фамилия Имя Отчество

В качестве разделителей могут выступать пробелы, табуляции, переводы строки.

Пример файла:

Иванов Василий Иванович

Сидоров Александр Михайлович

…

Метод readPersons должен возвращать стек.

Метод writePersons должен записывать в поток из стека (стек передается аргументом) информацию о людях в соответствии с вышеописанным форматом. Передаваемый методу writePersons стек не должен изменяться.

Класс PersonKeeper должен быть реализован в соответствии с шаблоном Singleton.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**2. Предлагаемое решение. Предлагаемое решение излагается на уровне идеи. Описывается общее решение задачи, с указанием структуры решения, в частности описание предлагаемых сущностей(классы), функций. Если в качестве решения предлагается архитектура ПО, она также описывается и представляется с помощью UML диаграммы. Если предложенная архитектура ПО базируется на одном их известных паттернов, то в отчете обязательно описывается назначение этого паттерна его архитектура, а также какие шаги были выполнены для применения этого паттерна для решения задачи. (Таким образом сдаете теоретическую часть, рассматриваемой темы + ваше решение)**

2.1

По вышесказанному заданию, нужно было реализовать класс в виде шаблона. Шаблонный класс позволяет оперировать данными разных типов, в чем и заключается основное отличие от простого класса.

Так же нужно было реализовать стек и работу с ним. Стек такая структура данных, которая представляется в виде упорядоченного списка, который будет работать по принципу LIFO (last-in-first-out), то есть последний помещённый элемент будет изъят первым. В качестве упорядоченного списка использовался однонаправленный список, который состоит из указателя на предыдущий узел и значения текущего узла.

Исключения обрабатываются классом EStackEmpty.

За помещение объекта в стек отвечает метод void push (const T &val). На вход подается значение, которое будет помещено в стек. Для начала создается новый узел, если узел не создался, то будет вызвано исключение. Созданный узел будет иметь переданное значение, а также будет являться последним элементом стека с указателем на ранее последний узел. Для отслеживания размера стека используется счетчик size, который при успешном добавлении элемента в стек, будет увеличиваться на один.

За извлечения объекта из стека отвечает метод const T pop (). Данный метод возвращает константное значение последнего узла. Если стек не имеет элементов, то будет вызвано исключение. Сначала сохраняется значение последнего узла, после чего сохраняется сам последний узел, а предыдущий узел становится последним, в конце освобождается память из под последнего сохраненного узла, уменьшается размер стека size и возвращается сохраненное значение.

За перебор значение стека отвечает метод void enumeration(std::function<void(const T &val)> func) const. Аргументом данной функции, является функция обратной связи, которая служит выводом значений стека. Метод проходится по всем элементам стека начинай с последнего, для каждого значения узла будет вызвана функция обратной связи.

За удаления всех узлов стека отвечает деструктор. Сохраняется последний узел, после чего предпоследний становится последним и затем удаляется последний сохраненный узел.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2.2

Также для решения поставленной задачи нужно использовать паттерн Singleton. Его суть заключается в том, что класс может быть только в одном экземпляре, а его конструктор, конструктор копирования и оператор присваивания находятся в приватной части класса. Для получения экземпляра класса используется метод static type instance(). UML диаграмма данного класса представлен на рисунке 1.

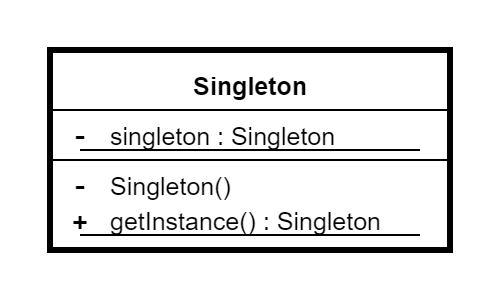


Рисунок 1 – класс Singleton

Для работы с ФИО людей реализован класс Person, который содержит ФИО, для получения и установки ФИО используются соответствующие get и set методы. Конструктор получает на вход строку с ФИО, разделение строки на составные части происходит благодаря методу split(). Данный метод принимает два параметра QRegularExpression("\\s+") и QString::SkipEmptyParts. Первый параметр отвечает за способ разделения (пробелы, табуляция), а второй нужен для того, что метод игнорировал пустые строки. Если строку не получилось разбить как необходимо, то будет вызвано исключение.

Экземпляры класса Person также будут храниться в стеке.

Класс PersonKeeper реализован согласно паттерну Singltone и содержит методы для чтения данных из файла и возвращает считанные данные и записи переданных данных в файл, а также возвращает единственный экземпляр Stack<Person>.

Параметром для метода чтения является путь к файлу и возвращает считанные данные в виде Stack<Person>, а для метода записи является также путь к файлу и Stack<Person>, в который будет записаны считанные данные. Оба метода проверяют открылся ли тот или иной файл, если нет, то выбрасывается исключение. Для чтения из файла используется метод readLineInto класса QTextStream, который построчно считывает данные в буферную строку, после чего в Stack<Person> через конструктор помещается считанная строка, которая обрабатывается вышеуказанным методом split(). Для записи в файл переданный стек перебирается функцией enumeration и у каждого объекта класса Person вызывается функция для получения данных либо через отдельные методы get для имени, фамилии и отчества, либо через общий get, который вернет полностью строку с ФИО. После успешного чтения и записи файлы закрываются.

Общая UML диграмма представленя на рисунке 2.

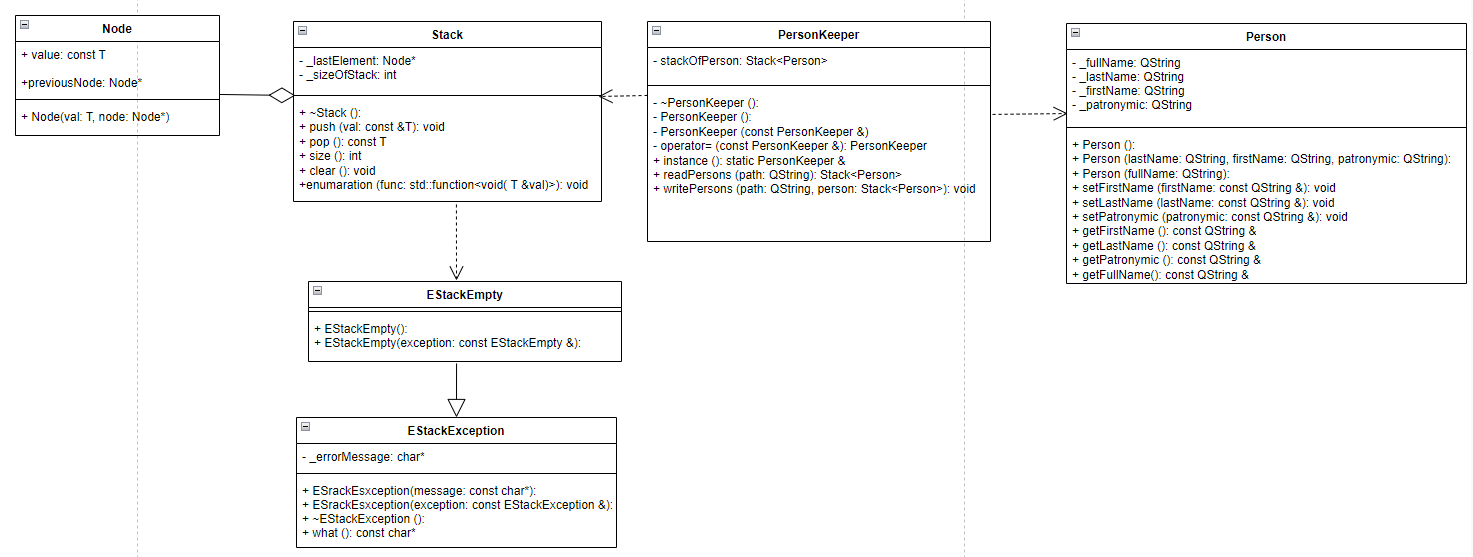


Рисунок 2 – общая UML диаграмма программы.

**3. Коды программ. полностью прокомментированы, с включением объяснений решения (фактически это ваш устный ответ).**

3.1

Программа содержит следящие файлы с подробными комментариями:

estackempty.h

exception.h

person.cpp

person.h

Stack.h

personkeeper.cpp

personkeeper.h

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**4. Инструкция пользователя. В этом разделе демонстрируются варианты взаимодействия с вашим ПО с точки зрения пользователя. Если требуется, то предоставляете готовые примеры (входные файлы с исходными данными и т.д.).**

4.1

Для того, чтобы создать объект нашего шаблонного класса используем следующую конструкцию:

Stack<тип данных> stack

Для добавления элемента в стек используется следующий метод:

stack.push(<тип данных> value);

Для того, чтобы достать из стека последний элемент используется следующий метод:

stack.pop();

Для того, чтобы просмотреть все элементы стека можно воспользоваться следующей конструкцией:

stack.enumaration([&](const <тип данных> &value) { std::cout << value << std::endl; }

Узнать размера стека можно следующим образом:

std::cout << stack.size();

Очистить весь стек:

stack.clear();

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

4.2

Для того, чтобы воспользоваться записью и чтением данных, нужно создать объект PersonKeeper:

PersonKeeper &personKeeper(PersonKeeper::instance());

Далее для чтения следуют создать объект Stack<Person> и задать имя/путь файла:

QString input = “input.txt”;

Stack<Person> person;

Затем у объекта personKeeper вызвать соответствующий метод:

person = personKeeper.readPersons(input);

Для записи нужно также задать имя/путь файла, после чего воспользоваться выше созданным Stack<Person> или создать свой новый и вызвать у personKeeper метод для записи:

QString output = “output.txt”;

personKeeper.writePersons(output, person);

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**5. Тестирование. В этом разделе описывается процесс тестирования, в частности все случаи (case1, case2) допустимые и недопустимые при которых ваше ПО выдает адекватную реакцию. Также, тестовые примеры, относительно которых проводилось тестирование.**

5.1

Не отходя далеко от темы чтения и записи данных в файл, продемонстрирую как происходит это при помощи кода программы в файле main.cpp:

QString inputFile = "input.txt"; // файл для чтения

QString outputFile = "output.txt"; // файл для записи

Stack<Person> person;

PersonKeeper &persorKeep(PersonKeeper::instance()); // экземпляр класса для чтения и записи данных

qDebug() << "Creat 2 files in the folder where the project exists: input.txt and output.txt";

// считываем имена из файла

qDebug() << "Read data from input.txt";

person = persorKeep.readPersons(inputFile);

qDebug() << "Data from input.txt: ";

// перебор стека для вывода

person.enumeration([&](const Person &value){

qDebug() << value.getLastName() << value.getFirstName() << value.getPatronymic();

});

qDebug() << "Write data from input.txt into ouput.txt";

// записываем имена в файл

persorKeep.writePersons(outputFile, person);

qDebug() << "Now you can open files to compare data";

Файл для чтения (“input.txt”) имеет следующие данные:

Melnikova Ksenia Vitalievna

Ivanova Sofia Ivanovna

Burakshaeva Yulia Sergeevna

Fursova Elizaveta Vladimirovna

Sapsay Ivan Alekseevich

В результате считывания и записи (“output.txt”) в файл получится следующий файл с записями:

Sapsay Ivan Alekseevich

Fursova Elizaveta Vladimirovna

Burakshaeva Yulia Sergeevna

Ivanova Sofia Ivanovna

Melnikova Ksenia Vitalievna

Если же данные не будут соответствующим условиям, то вызовется исключение.

Пример не корректных данных:

“Alla”

“Alla Melnikova Ivanova Olya”

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

5.2

Далее проведены тесты для int и float:

Stack<int> stackOfInteger; // создаем стек с int

Stack<float> stackOffloat; // создаем стек с float

qDebug() << "Add 10 randoms values of integer into stackOfInteger";

for(int i = 0; i < 10; ++i) {

int val = rand() % 10 + 1;

stackOfInteger.push(val); // заполняем стек псевдо рандомными значениями тип int

qDebug() << val;

}

qDebug() << "Output the stackOfInteger: ";

stackOfInteger.enumeration([&](const int &value){

qDebug() << value; // выводим в консоль получившийся стек из int

});

qDebug() << "Add 10 values of float into stackOfFloat";

for(int i = 0; i < 10; ++i) {

float val = static\_cast<float>(i / (i + 2.5)); // заполняем стек псевдо рандомными значениями тип float

stackOffloat.push(val);

qDebug() << val;

}

qDebug() << "Output the stackOfFloat: ";

stackOffloat.enumeration([&](const float &value){

qDebug() << value; // выводим в консоль получившийся стек из float

});

Так же можно узнать размер стека:

qDebug() << stackOfInteger.size();

Последним тестированием будет извлечение данных из пустого стека типа bool:

Stack<bool> stack;

stack.pop();

Программа вызовет исключение, что стек пуст.