Реализация шаблонного класса стек. Запрограммируйте шаблонный класс, реализующий стек.

1. Постановка задачи

Реализация шаблонного класса стек. Запрограммируйте шаблонный класс, реализующий стек. Класс должен поддерживать следующие операции:

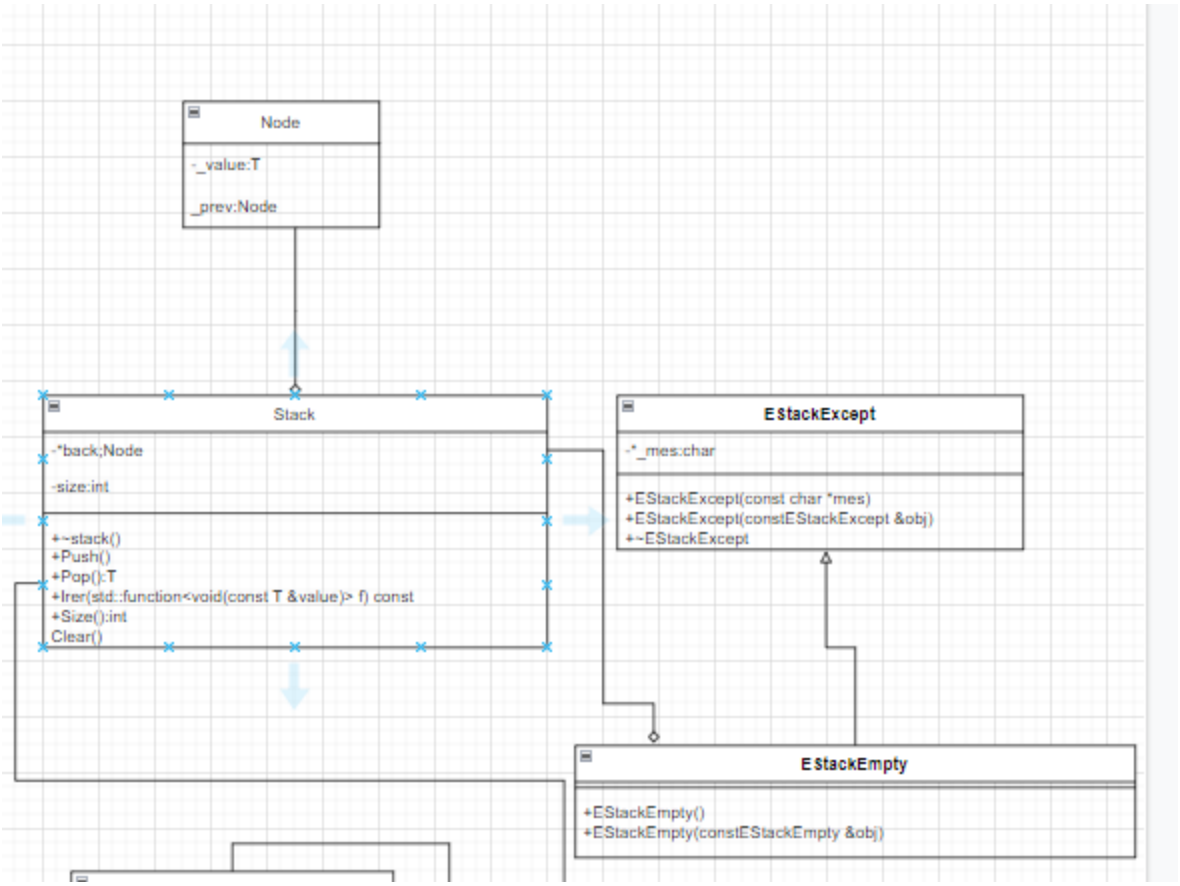
1. Помещение объекта в стек;
2. Извлечение объекта из стека;
3. Получение размерности стека.

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.

2. Предлагаемое решение.

Из поставленной следует ,что класс стек будет реализован через шаблонный класс. Поэтому стоит учесть ,что разбиении данного класса на 2 файла не будут видны методы,следовательно лучше реализовать всё в одном файле(stack.h).
Способ реализации стека я выбрал через односвязный список,потому что заготовки по данному реализации уже были у меня.Звено стека состоит из двух значении,value-значения переменной и prev указатель на предыдущий элемент стека.
Реализуемые методы:

1. Void Push (const T &value) (Добавления значения в конец списка).
   Суть метода заключается в том ,что создать новое звено и добавить его в конец ,с учетом проверки созданного звена.Так же не забываем увеличить размерность на 1 .
2. Void Clear() (Очистка стека)
   Удаления объектов путем всей итерации стека и удаления элементов, пока последнее звено не станет пустым
3. Десткуртор таже идея ,что и Clear()
4. Int Size()
   Возвращение переменную size
5. const T Pop()(Извлечения последнего объекта из стека)
   Извлечения объекта из стека путем сохранения данного значения в переменную с последующим удаления данного звена из стека с учетом проверки пустоты стека и возвращения данного значения .Также надо уменьшить размерность стека на 1
6. void Iter(std::function<void(const T &value)> f) const (Переборка элементов стека)
   Сначала идея этой функции был перебор и вывод значении стека,но для дальнейшей работы было переписана на итерацию на лямбда функцию
   Исключения, как и было предложено, реализованы в виде двух классов. Общий класс исключений стека и производный от него - исключение переполнения



. UML

3 Коды программ

К данному этапу в проекте относятся следующие файлы:

1. stack.h
2. EStackEmpthy.h
3. EstackExcept.h

4. Инструкция пользователя.

Данная реализация является шаблонным классом ,поэтому при создании стека надо указать тип данных .В моем примере решил указать int
В начале программы инициализируем стек нужного типа
Пример работы на int
stack<int> kek;
Для заполнения его данными используете метод push
kek.Push(1);
kek.Push(2);
kek.Push(3);
Также пользователь может извлечь значения последнего звена стека
kek.Pop();
Также рассмотрим функцию о размерности стека

kek.Size();
Возвращение размера стека

5. Тестирование.

Рассмотрим несколько исключении

```
 stack<int> kek;
   kek.Pop();
```

Данная программа выбросит исключение о том, что стек пуст

```
while (true)
{
        cout << kek.Pop() << endl;
}
```

В данной программе нет условия выхода из цикла, но благодаря тому что у нас на данный случай предусмотрено исключение, пользователь получит удобное предупреждение, а не сообщение об ошибке.

Реализация класса PersonKeeper

1. Постановка задачи

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:

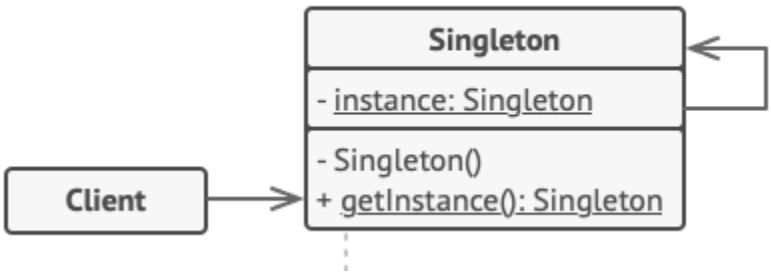
Иванов Василий Иванович
Сидоров Александр Михайлович
…

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.

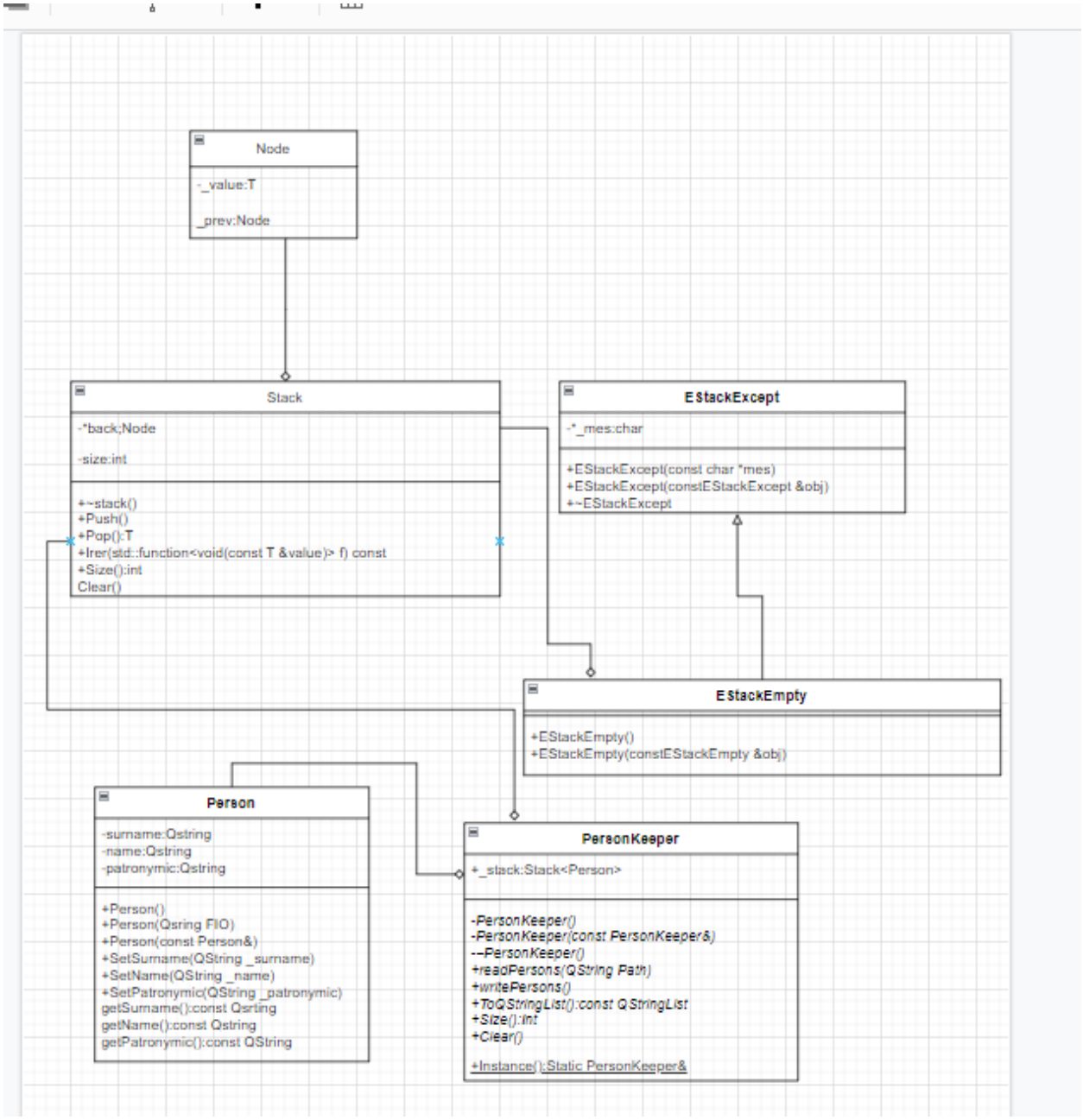
2. Предлагаемое решение.

Для начало рассмотрим шаблон одиночки(Singleton)
Одиночка — это порождающий паттерн проектирования, который гарантирует, что у класса есть только один экземпляр, и предоставляет к нему глобальную точку доступа.
В нашем случае это достигается с помощью конструктора по умолчанию и копирования в приватную область.Чтобы получить доступ к ед экземпляру используется статический метод Instance().Его суть заключается в создается статическая переменная ,которая создается только при первом вызове ,а в последующих будет просто возвращаться ссылка на созданный ранее экземпляр.
UML пример :



В начале нужно создать класс в которых будут реализованы сами личности.В начале я решил реализовать данный класс через обычный string .Наше имя будет разбито на 3 сегмента имя,фамилия,отчества.Также для получения нужных данных были написаны гетары и сетары и два конструктора.Один по умолчанию ,второй по считыванию полной строки.Так как полный строку нужно разбит на 3 сегмента,нужно функция ,которая будет сплитить строку.Так как в начале разработки я использовал обычную строку функция сплита различалась .В начале функция сплита реализовалась через поиск отступа последующем деления строки и присвоения сегмента в нужные нам элементы,но у данного подхода было несколько минусов.Он конфликтовал с qt и если строка состояла больше чем и 3 сегментов он ломался.Так что с этого момента произошла резкая смена с обычного стринга на qt аналог.Данное действия поняла переменные гетера и сетера и сменила идею дробления строки с учетом прошлых ошибок.Сплит в этот раз происходил с помощью создания массива строк ,которые сплитились с помощью regex с учетом ,что слов в строке подходящая количество.На этом рассмотрения класса person закончилось,можно перейти к keeper.
По заданию данные должны храниться в стеке.В условия сказано,что класс должен выполнять readPersons и readPersons:
1)void readPersons(QString path)(чтения данных из файла)
Чтения данных из файла в Qtext с последующим считыванием его по линиям и добавлением их в стек с учетом проверке ,что файл открывался. Не забывая закрыть файл.
2)void writePersons(QString path) const (запись данных в файл)
Запись данных файл путем итерации стека и получение данных через гетары с учетом

проверке ,что файл открывался. Не забывая закрыть файл.



UML Diagramma

3. Коды программ.

person.h
person.ccp
personKeeper.h
personKeeper.ccp
Stack.h

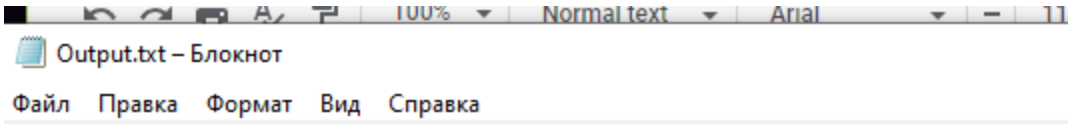
4. Инструкция пользователя.

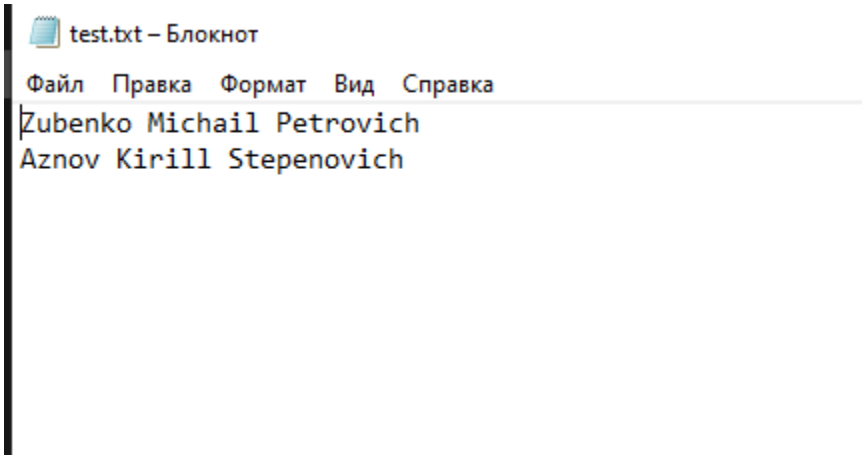
Так как реализация шла через одиночек ,все действия пользователя будут происходит через статический метод PersonKeeper::Instance() для получения экземпляра класса. Пользователь может получить данные через функция readPersons(ваш путь к файлу) указав полный путь к файлу .Соответственно, данная функция пополняет хранилище данными из файла. Следует отметить, что перед этим она не удаляет имеющиеся данные.
Второй по важности функции является writePersons(ваш путь к файлу) запись ваших данных указанный вами файл.Данная функция не изменяет хранящиеся в хранилище данные.
Size() -показывает количество ваших личностей
Clear() Очистка хранилища

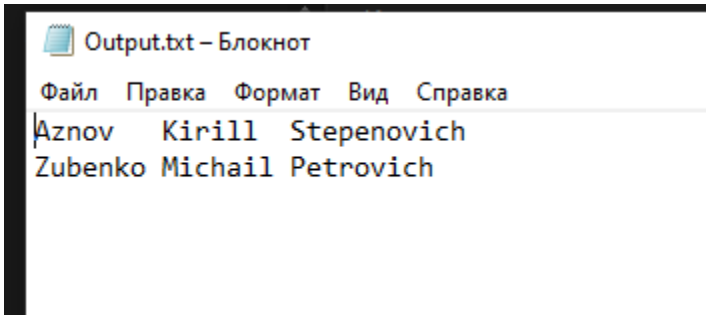
5. Тестирование.

Дефолтные действия с данным классом это забрать данные с одного файла и перекинуть его в другой

```
PersonKeeper::Instance().readPersons("C:/Users/andru/Documents/Lab1/test.txt");
PersonKeeper::Instance().writePersons("C:/Users/andru/Documents/Lab1/Output.txt");;
```

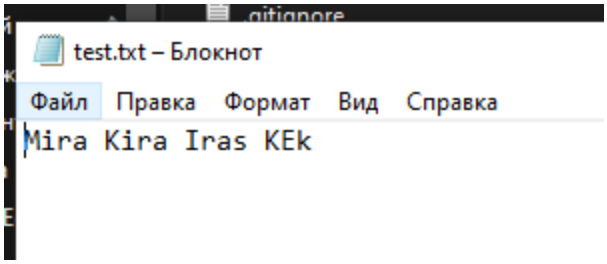


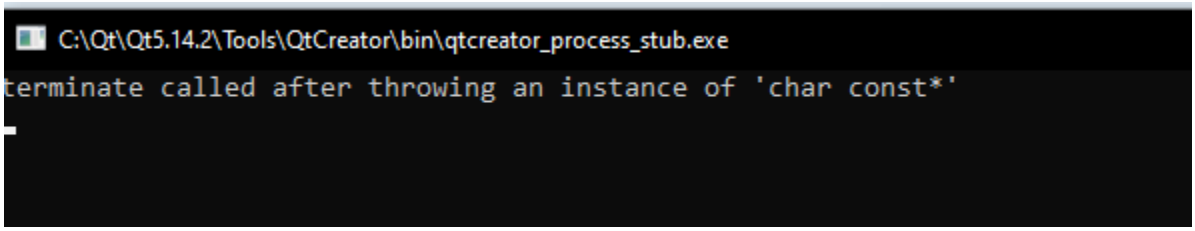




Все работает ,из-за того что в реализации используется регулярные выражения количество пробелов роли не влияет
Теперь идем к изменения строк ввода
Рассмотрим несколько случаев:

1. Ввод больше 3 слов(ловим исключения про формат персон или двух)


.gitignore
test.txt – Блокнот
Файл Правка Формат Вид Справка
Mira Kira Iras KEk


C:\Qt\Qt5.14.2\Tools\QtCreator\bin\qtcreator_process_stub.exe
terminate called after throwing an instance of 'char const*'