Министерство науки и высшего образования Российской Федерации
Калужский филиал
федерального государственного бюджетного
образовательного учреждения высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(КФ МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ***ИУК «Информатика и управление»***

КАФЕДРА ***ИУК4 «Программное обеспечение ЭВМ, информационные технологии»***

ЛАБОРАТОРНАЯ РАБОТА №4

«Знакомство с контейнерами»

ДИСЦИПЛИНА: «Высокоуровневое программирование»

Выполнил: студент гр. ИУК4-22Б _____ (Карельский М.К.)
(Подпись) (Ф.И.О.)

Проверил: _____ (Козина А.В.)
(Подпись) (Ф.И.О.)

Дата сдачи (защиты):

Результаты сдачи (защиты):

- Балльная оценка:
- Оценка:

Калуга , 2021

Цель: приобретение практических навыков и знаний по созданию и обработки классов – контейнеров данных.

Задачи:

1. Изучить понятие контейнера;
2. Научиться описывать простой контейнер класса;
3. Изучить написание элементов для контейнера;
4. Познакомиться с умными указателями;
5. Познакомиться с итераторами и научиться применять их;
6. Реализовать свою структуру контейнера.

Вариант 8

Задание:

Общее задание:

Задача 1

Создайте библиотеку (папку в корне вашего проекта, а в ней файлы), которая будет являться неким глобальным хранилищем данных вашей программы, назовите её Store. В ней создайте структуру данных State (напишите два файла State.h и State.cpp), в заголовочном файле этой структуры пропишите поля, в которых будут храниться наборы ваших сущностей (по одному полю для каждого типа набора) для реализации наборов можете использовать библиотеку STL, а конкретно тип: vector. Причём данные, которые будет хранить вектор должны быть указателями на объекты сущностей.

Задача 2

После того, как вы стали уверенно чувствовать себя при работе с классами, мы можем упростить нашу архитектуру программы. По факту наши сущности – это просто объекты с данными, которые не хранят в себе никаких методов по работе с ними. В таких случаях не применяется ООП, а используется обычная структура. Поэтому перепишите те сущности, в которых не используется ООП на обычные структуры, удалив геттеры/сеттеры, а также убрав модификаторы доступа. (если вы использовали для полей префикс _m, в структурах его следует убрать). Стоит отметить, что если в сущностях используется наследование, то это уже ООП, такие сущности следует реализовывать только через классы. Также, если сеттер отвечает за обработку корректного значения ввода, то также оставляем сущность классом.

Задача 3

Перепишите все поля, которые использовали с – строки, на использование класса string (можно из STL библиотеки, а можно из методических указаний).

Задача 4

Создайте библиотеку экранов (папку в корне своего проекта под названием Screens, в которой будут лежать файлы). В этой библиотеки создайте под директории, в которых уже будут находиться сами экраны. Каждый экран будет

представлять из себя класс, который наследуется от интерфейса InterfaceScreen (который будет лежать в корне папки с экранами). В интерфейсе будет две виртуальные чистые функции: int start(int) и void renderMain() const. Первая будет являться точкой входа в экран, а вторая будет отвечать за базовую его отрисовку в консоли. Каждый экран будет переопределять эти методы по своему усмотрению. Помимо переопределённых методов в экранах, будут и частные методы, которые относятся непосредственно к самому функционалу экрана, например: показ списка пользователей, сортировка, добавление, удаление и т д. Экран – это по сути логическая область вашей программы, которая будет отвечать за тот, или иной раздел функциональности. Разбейте вашу программу на логические блоки (по сути, она уже почти разбита в индивидуальном задании) – экраны и реализуйте в них частную функциональность. Пока, в качестве заглушки, включите в интерфейсный экран библиотеку Store и используйте объекты State, чтобы управлять данными.

Основная задача:

Вам будет предложено написать программу – «Автоматизированная система диалога (чат бот)». Которая будет включать следующий функционал:

- Ведение базы пользователей
  - Создание / удаление / редактирование записей
  - Сортировка / фильтрация
- Ведение базы диалогов, тем, интересов и напоминаний
- Возможность авторизации
- Создание файлов-отчётов и сохранения состояния

UML-диаграмма классов:

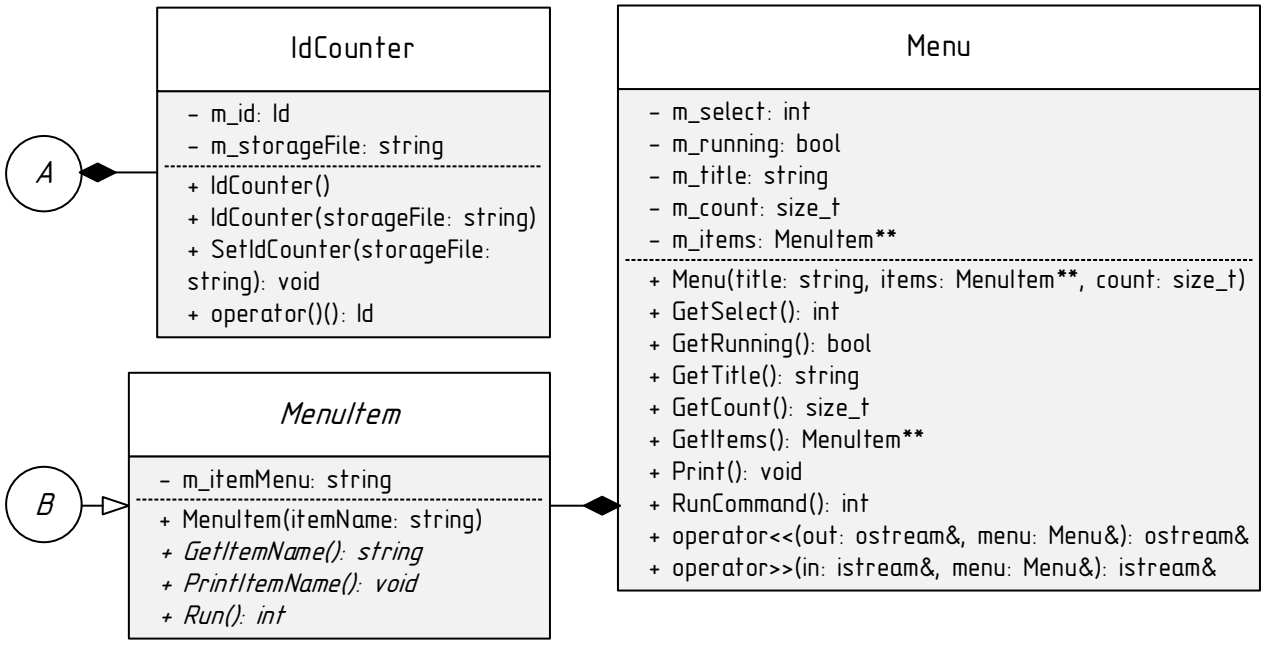

Рисунок 1.1. UML-диаграмма классов

Рисунок 1.2. UML-диаграмма классов

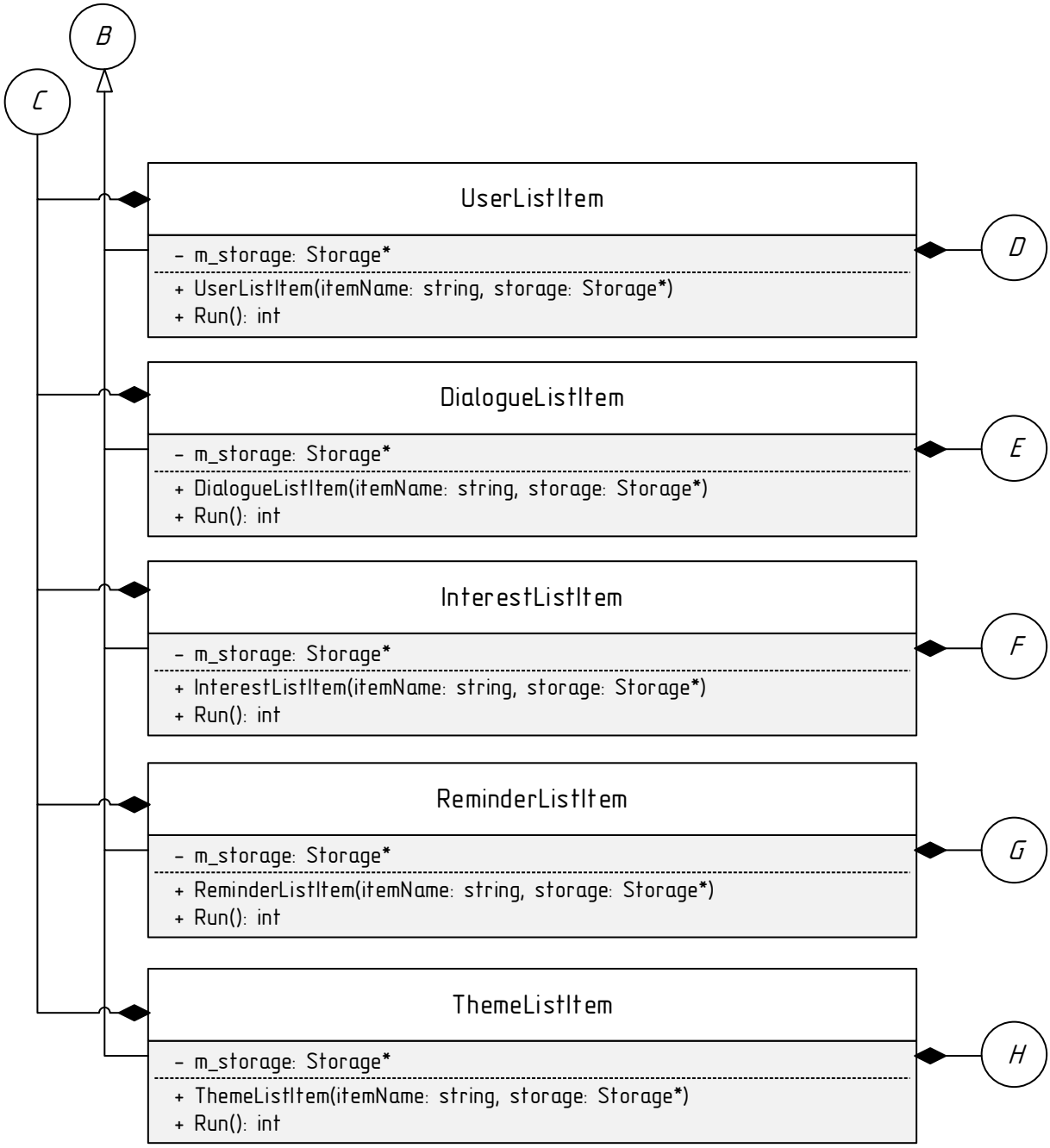

Рисунок 1.3. UML-диаграмма классов

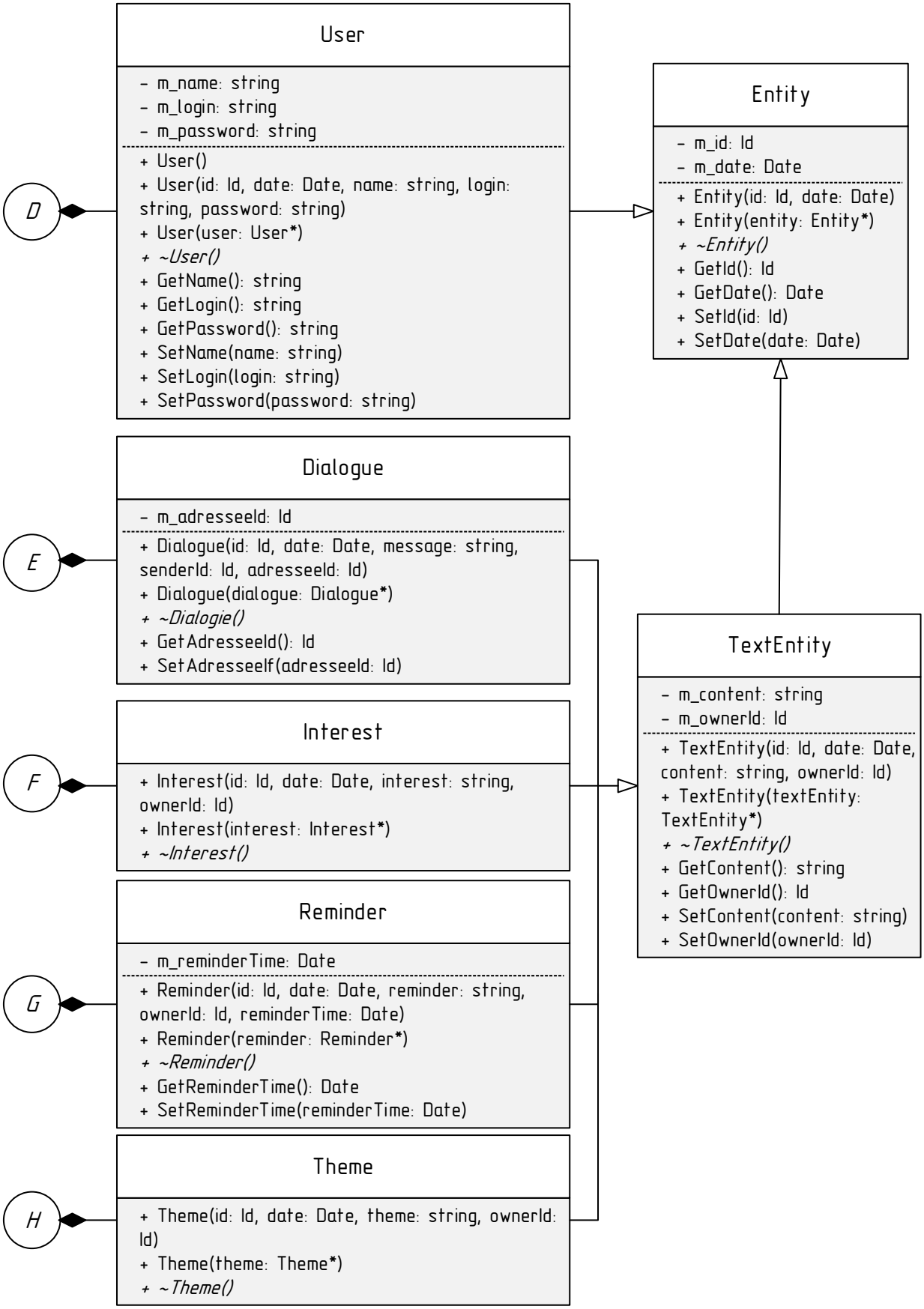

Рисунок 1.4. UML-диаграмма классов

Листинг:
DialogueListItem.h

```
#ifndef DIALOGUE_LIST_ITEM_H
#define DIALOGUE_LIST_ITEM_H
#include "AbstractMenuItem.h"
#include "Dialogue.h"
#include "Storage.h"

namespace KMK
{
	class DialogueListItem :
public MenuItem
	{
	public:

	DialogueListItem(std::strin
g itemName, Storage* storage);

		int Run();

	private:
		Storage* m_storage{};
	};
}

#endif // !DIALOGUE_LIST_ITEM_H
```

DialogueListItem.cpp

```
#include "DialogueListItem.h"
#include <iostream>
#include <fstream>
#include "Add.h"
#include <Windows.h>
#include "Remove.h"
#include "Edit.h"
#include "Sort.h"
#include "Filter.h"
#include <iomanip>
#include <string>

using namespace KMK;

DialogueListItem::DialogueListIt
em(std::string itemName,
Storage* storage) :
	MenuItem(itemName)
{
	m_storage = storage;
}

int DialogueListItem::Run()
{
	enum  Command
	{
		RESET,
		ADD,
		REMOVE,
		EDIT,
		SORT,
		FILTER,
		ID,
		EXIT
	};

	unsigned short command = 0;

	while (command != EXIT)
	{
		size_t size =
m_storage-
>GetDialogueListSize();

		unsigned short
maximumMessageLength = 7;
		for (Iteration i{}; i
< size; ++i)
		{
			if (m_storage-
>m_dialogueList[i]-
>GetContent().length() >
maximumMessageLength)
			{

	maximumMessageLength =
m_storage->m_dialogueList[i]-
>GetContent().length();
			}
		}

		std::cout <<
std::setw((11 + 3 + 3 + 5 +
maximumMessageLength + 1 + 11 +
11 + 6 + GetItemName().length())
/ 2) << GetItemName() << "\n\n";
		std::cout <<
std::setw(11) << "ID" << "|";
		std::cout <<
std::setw(3) << "dd" << "|";
		std::cout <<
std::setw(3) << "mm" << "|";
		std::cout <<
std::setw(5) << "yyyy" << "|";
		std::cout <<
std::setw(maximumMessageLength +
1) << "Message" << "|";
```

```
            std::cout <<
std::setw(11) << "Sender ID" <<
"|";
            std::cout <<
std::setw(11) << "Adressee ID";
            std::cout << '\n';

            for (Iteration i{}; i
< size; ++i)
            {
                std::cout <<
std::setw(11) << m_storage-
>m_dialogueList[i]->GetId() <<
"|";
                std::cout <<
std::setw(3) << m_storage-
>m_dialogueList[i]-
>GetDate().day << "|";
                std::cout <<
std::setw(3) << m_storage-
>m_dialogueList[i]-
>GetDate().month << "|";
                std::cout <<
std::setw(5) << m_storage-
>m_dialogueList[i]-
>GetDate().year << "|";
                std::cout <<
std::setw(maximumMessageLength +
1) << m_storage-
>m_dialogueList[i]->GetContent()
<< "|";
                std::cout <<
std::setw(11) << m_storage-
>m_dialogueList[i]->GetOwnerId()
<< "|";
                std::cout <<
std::setw(11) << m_storage-
>m_dialogueList[i]-
>GetAdresseeId();
                std::cout <<
'\n';
            }

            std::cout << '\n';
            std::cout << RESET <<
". Reset list\n";
            std::cout << ADD << ".
Add new dialogue\n";
            std::cout << REMOVE <<
". Delete dialogue\n";
            std::cout << EDIT <<
". Edit dialogue\n";
            std::cout << SORT <<
". Sort list\n";
            std::cout << FILTER <<
". Filter list\n";
            std::cout << ID << ".
Choose ID\n";
            std::cout << EXIT <<
". Exit\n";
            std::cout << "Input
command: ";
            std::cin >> command;
            std::cin.ignore();
            std::cout << '\n';

            if (command == RESET)
            {
                m_storage-
>LoadDialogueList();
            }
            else if (command ==
ADD)
            {
                std::string
message{};
                std::cout <<
"Input message: ";

    std::getline(std::cin,
message);

                Id senderId;
                std::cout <<
"Input sender ID: ";
                std::cin >>
senderId;

                Id adresseeId;
                std::cout <<
"Input adressee ID: ";
                std::cin >>
adresseeId;

    std::cin.ignore();

                SYSTEMTIME
systemTime;

    GetLocalTime(&systemTime);
                Dialogue*
newDialogue = new
Dialogue(m_storage-
>GetNextDialogueId(), {
systemTime.wDay,
systemTime.wMonth,
systemTime.wYear }, message,
senderId, adresseeId);
```

```
     Add((Entity**&)m_storage-
>m_dialogueList, size,
newDialogue);
               m_storage-
>SetDialogueListSize(size);
          }
          else if (command ==
REMOVE)
          {
               std::cout <<
"Input ID: ";
               Id id;
               std::cin >> id;

     std::cin.ignore();


     Remove((Entity**&)m_storage
->m_dialogueList, size, id);
               m_storage-
>SetDialogueListSize(size);
          }
          else if (command ==
EDIT)
          {
               std::cout <<
"Input ID: ";
               Id id;
               std::cin >> id;

     std::cin.ignore();
               std::cout <<
'\n';

               std::cout <<
"Fields to edit\n";
               std::cout << "0.
ID\n";
               std::cout << "1.
Date\n";
               std::cout << "2.
Message\n";
               std::cout << "3.
Sender ID\n";
               std::cout << "4.
Adressee ID\n";
               std::cout <<
"Choose field: ";
               unsigned short
fieldToChange;
               std::cin >>
fieldToChange;

     std::cin.ignore();
               std::cout <<
'\n';

               if (fieldToChange
== 0)
               {
                    std::cout <<
"Input new ID: ";
                    Id* newId =
new Id{};
                    std::cin >>
*newId;

     std::cin.ignore();

     Edit((Entity**&)m_storage-
>m_dialogueList, size, id,
(void*)newId, FieldMode::ID);
                    delete
newId;
               }
               if (fieldToChange
== 1)
               {
                    std::cout <<
"Input new date\n";
                    std::cout <<
"Day:";
                    unsigned
short day;
                    std::cin >>
day;
                    std::cout <<
"Month:";
                    unsigned
short month;
                    std::cin >>
month;
                    std::cout <<
"Year:";
                    unsigned
short year;
                    std::cin >>
year;

     std::cin.ignore();

     Entity::Date* newDate = new
Entity::Date{ day, month, year
};

     Edit((Entity**&)m_storage-
>m_dialogueList, size, id,
(void*)newDate,
FieldMode::DATE);
```

```
                    delete
newDate;
               }
               if (fieldToChange
== 2)
               {
                    std::cout <<
"Input new message: ";
                    std::string*
message = new std::string{};

     std::getline(std::cin,
*message);

     Edit((Entity**&)m_storage-
>m_dialogueList, size, id,
(void*)message,
FieldMode::CONTENT);
                    delete
message;
               }
               if (fieldToChange
== 3)
               {
                    std::cout <<
"Input new sender ID: ";
                    Id*
newSenderId = new Id{};
                    std::cin >>
*newSenderId;

     std::cin.ignore();

     Edit((Entity**&)m_storage-
>m_dialogueList, size, id,
(void*)newSenderId,
FieldMode::OWNER_ID);
                    delete
newSenderId;
               }
               if (fieldToChange
== 4)
               {
                    std::cout <<
"Input new adressee ID: ";
                    Id*
newAdresseeId = new Id{};
                    std::cin >>
*newAdresseeId;

     std::cin.ignore();

     Edit((Entity**&)m_storage-
>m_dialogueList, size, id,
(void*)newAdresseeId,
FieldMode::OWNER_ID);
                    delete
newAdresseeId;
               }
          }
          else if (command ==
SORT)
          {
               std::cout <<
"Orders for sort\n";
               std::cout << "0.
Descending\n";
               std::cout << "1.
Ascending\n";
               std::cout <<
"Choose order: ";
               unsigned short
order;
               std::cin >>
order;
               std::cout <<
'\n';

               std::cout <<
"Fields for sort\n";
               std::cout << "0.
ID\n";
               std::cout << "1.
Date\n";
               std::cout << "2.
Message\n";
               std::cout << "3.
Sender ID\n";
               std::cout << "4.
Adressee ID\n";
               std::cout <<
"Choose field: ";
               unsigned short
field;
               std::cin >>
field;

     std::cin.ignore();
               FieldMode
sortMode = (FieldMode)-1;
               switch (field)
               {
               case 0:
                    sortMode =
FieldMode::ID;
                    break;
               case 1:
                    sortMode =
FieldMode::DATE;
```

```
                    break;
               case 2:
                    sortMode =
FieldMode::CONTENT;
                    break;
               case 3:
                    sortMode =
FieldMode::OWNER_ID;
                    break;
               case 4:
                    sortMode =
FieldMode::ADRESSEE_ID;
                    break;
               }

     Sort((Entity**&)m_storage-
>m_dialogueList, size,
(OrderMode)order, sortMode);
          }
          else if (command ==
FILTER)
          {
               std::cout <<
"Fields for filter\n";
               std::cout << "0.
ID\n";
               std::cout << "1.
Date\n";
               std::cout << "2.
Message\n";
               std::cout << "3.
Sender ID\n";
               std::cout << "4.
Adressee ID\n";
               std::cout <<
"Choose field: ";
               unsigned short
field;
               std::cin >>
field;

     std::cin.ignore();
               std::cout <<
'\n';

               if (field == 0)
               {
                    std::cout <<
"Input part of ID: ";
                    Id id;
                    std::cin >>
id;

     std::cin.ignore();

     Filter((Entity**&)m_storage
->m_dialogueList, size, id,
FieldMode::ID);
               }
               if (field == 1)
               {
                    std::cout <<
"Input date (if you don't want
to filter by the field, input
0)\n";
                    std::cout <<
"Day: ";
                    unsigned
short day;
                    std::cin >>
day;
                    std::cout <<
"Month: ";
                    unsigned
short month;
                    std::cin >>
month;
                    std::cout <<
"Year: ";
                    unsigned
short year;
                    std::cin >>
year;

     std::cin.ignore();
                    Entity::Date
date{ day, month, year };

     Filter((Entity**&)m_storage
->m_dialogueList, size, date,
FieldMode::DATE);
               }
               if (field == 2)
               {
                    std::cout <<
"Input part of message: ";
                    std::string
message{};

     std::getline(std::cin,
message);

     Filter((Entity**&)m_storage
->m_dialogueList, size, message,
FieldMode::CONTENT);
               }
               if (field == 3)
               {
```

```
                    std::cout <<
"Input part of sender ID: ";
                    Id senderId;
                    std::cin >>
senderId;

     std::cin.ignore();

     Filter((Entity**&)m_storage
->m_dialogueList, size,
senderId, FieldMode::OWNER_ID);
               }
               if (field == 4)
               {
                    std::cout <<
"Input part of adressee ID: ";
                    Id
adresseeId;
                    std::cin >>
adresseeId;

     std::cin.ignore();

     Filter((Entity**&)m_storage
->m_dialogueList, size,
adresseeId,
FieldMode::ADRESSEE_ID);
               }
               m_storage-
>SetDialogueListSize(size);
          }

          if (command == ADD ||
command == REMOVE || command ==
EDIT)
          {
               m_storage-
>SaveDialogueList();
          }

          system("cls");
     }

     return 0;
}
```

InterestListItem.h

```
#ifndef INTEREST_LIST_ITEM_H
#define INTEREST_LIST_ITEM_H
#include "AbstractMenuItem.h"
#include "Interest.h"
#include "Storage.h"

namespace KMK
{
     class InterestListItem :
public MenuItem
     {
     public:

     InterestListItem(std::strin
g itemName, Storage* storage);

          int Run();

     private:
          Storage* m_storage{};
     };
}

#endif // !INTEREST_LIST_ITEM_H
```

InterestListItem.cpp

```
#include "InterestListItem.h"
#include <iostream>
#include <fstream>
#include "Add.h"
#include <Windows.h>
#include "Remove.h"
#include "Edit.h"
#include "Sort.h"
#include "Filter.h"
#include <iomanip>
#include <string>

using namespace KMK;

InterestListItem::InterestListIt
em(std::string itemName,
Storage* storage) :
     MenuItem(itemName)
{
     m_storage = storage;
}

int InterestListItem::Run()
{
     enum  Command
     {
          RESET,
          ADD,
          REMOVE,
          EDIT,
          SORT,
          FILTER,
          ID,
```

```
          EXIT
     };

     unsigned short command = 0;

     while (command != EXIT)
     {
          size_t size =
m_storage-
>GetInterestListSize();

          unsigned short
maximumInterestLength = 8;
          for (Iteration i{}; i
< size; ++i)
          {
               if (m_storage-
>m_interestList[i]-
>GetContent().length() >
maximumInterestLength)
               {

     maximumInterestLength =
m_storage->m_interestList[i]-
>GetContent().length();
               }
          }

          std::cout <<
std::setw((11 + 3 + 3 + 5 +
maximumInterestLength + 1 + 11 +
5 + GetItemName().length()) / 2)
<< GetItemName() << "\n\n";
          std::cout <<
std::setw(11) << "ID" << "|";
          std::cout <<
std::setw(3) << "dd" << "|";
          std::cout <<
std::setw(3) << "mm" << "|";
          std::cout <<
std::setw(5) << "yyyy" << "|";
          std::cout <<
std::setw(maximumInterestLength
+ 1) << "Interest" << "|";
          std::cout <<
std::setw(11) << "Owner ID";
          std::cout << '\n';

          for (Iteration i{}; i
< size; ++i)
          {
               std::cout <<
std::setw(11) << m_storage-
>m_interestList[i]->GetId() <<
"|";
               std::cout <<
std::setw(3) << m_storage-
>m_interestList[i]-
>GetDate().day << "|";
               std::cout <<
std::setw(3) << m_storage-
>m_interestList[i]-
>GetDate().month << "|";
               std::cout <<
std::setw(5) << m_storage-
>m_interestList[i]-
>GetDate().year << "|";
               std::cout <<
std::setw(maximumInterestLength
+ 1) << m_storage-
>m_interestList[i]->GetContent()
<< "|";
               std::cout <<
std::setw(11) << m_storage-
>m_interestList[i]-
>GetOwnerId();
               std::cout <<
'\n';
          }

          std::cout << '\n';
          std::cout << RESET <<
". Reset list\n";
          std::cout << ADD << ".
Add new interest\n";
          std::cout << REMOVE <<
". Delete interest\n";
          std::cout << EDIT <<
". Edit interest\n";
          std::cout << SORT <<
". Sort list\n";
          std::cout << FILTER <<
". Filter list\n";
          std::cout << ID << ".
Choose ID\n";
          std::cout << EXIT <<
". Exit\n";
          std::cout << "Input
command: ";
          std::cin >> command;
          std::cin.ignore();
          std::cout << '\n';

          if (command == RESET)
          {
               m_storage-
>LoadInterestList();
          }
          else if (command ==
ADD)
```

```
        {
            std::string
interest{};
            std::cout <<
"Input interest: ";

    std::getline(std::cin,
interest);

            Id ownerId;
            std::cout <<
"Input owner ID: ";
            std::cin >>
ownerId;

    std::cin.ignore();

            SYSTEMTIME
systemTime;

    GetLocalTime(&systemTime);
            Interest*
newInterest = new
Interest(m_storage-
>GetNextInterestId(), {
systemTime.wDay,
systemTime.wMonth,
systemTime.wYear }, interest,
ownerId);


    Add((Entity**&)m_storage-
>m_interestList, size,
newInterest);
            m_storage-
>SetInterestListSize(size);
        }
        else if (command ==
REMOVE)
        {
            std::cout <<
"Input ID: ";
            Id id;
            std::cin >> id;

    std::cin.ignore();


    Remove((Entity**&)m_storage
->m_interestList, size, id);
            m_storage-
>SetInterestListSize(size);
        }
        else if (command ==
EDIT)
        {
            std::cout <<
"Input ID: ";
            Id id;
            std::cin >> id;

    std::cin.ignore();
            std::cout <<
'\n';

            std::cout <<
"Fields to edit\n";
            std::cout << "0.
ID\n";
            std::cout << "1.
Date\n";
            std::cout << "2.
Interest\n";
            std::cout << "3.
Owner ID\n";
            std::cout <<
"Choose field: ";
            unsigned short
fieldToChange;
            std::cin >>
fieldToChange;

    std::cin.ignore();
            std::cout <<
'\n';

            if (fieldToChange
== 0)
            {
                std::cout <<
"Input new ID: ";
                Id* newId =
new Id{};
                std::cin >>
*newId;

    std::cin.ignore();

    Edit((Entity**&)m_storage-
>m_interestList, size, id,
(void*)newId, FieldMode::ID);
                delete
newId;
            }
            if (fieldToChange
== 1)
            {
                std::cout <<
"Input new date\n";
```

```
                    std::cout <<
"Day: ";
                    unsigned
short day;
                    std::cin >>
day;
                    std::cout <<
"Month: ";
                    unsigned
short month;
                    std::cin >>
month;
                    std::cout <<
"Year: ";
                    unsigned
short year;
                    std::cin >>
year;

     std::cin.ignore();

     Entity::Date* newDate = new
Entity::Date{ day, month, year
};

     Edit((Entity**&)m_storage-
>m_interestList, size, id,
(void*)newDate,
FieldMode::DATE);
                    delete
newDate;
               }
               if (fieldToChange
== 2)
               {
                    std::cout <<
"Input new interest: ";
                    std::string*
interest = new std::string{};

     std::getline(std::cin,
*interest);

     Edit((Entity**&)m_storage-
>m_interestList, size, id,
(void*)interest,
FieldMode::CONTENT);
                    delete
interest;
               }
               if (fieldToChange
== 3)
               {
                    std::cout <<
"Input new owner ID: ";
                    Id*
newOwnerId = new Id{};
                    std::cin >>
*newOwnerId;

     std::cin.ignore();

     Edit((Entity**&)m_storage-
>m_interestList, size, id,
(void*)newOwnerId,
FieldMode::OWNER_ID);
                    delete
newOwnerId;
               }
          }
          else if (command ==
SORT)
          {
               std::cout <<
"Orders for sort\n";
               std::cout << "0.
Descending\n";
               std::cout << "1.
Ascending\n";
               std::cout <<
"Choose order: ";
               unsigned short
order;
               std::cin >>
order;
               std::cout <<
'\n';

               std::cout <<
"Fields for sort\n";
               std::cout << "0.
ID\n";
               std::cout << "1.
Date\n";
               std::cout << "2.
Interest\n";
               std::cout << "3.
Owner ID\n";
               std::cout <<
"Choose field: ";
               unsigned short
field;
               std::cin >>
field;

     std::cin.ignore();
               FieldMode
sortMode = (FieldMode)-1;
               switch (field)
               {
```

```
                case 0:
                    sortMode =
FieldMode::ID;
                    break;
                case 1:
                    sortMode =
FieldMode::DATE;
                    break;
                case 2:
                    sortMode =
FieldMode::CONTENT;
                    break;
                case 3:
                    sortMode =
FieldMode::OWNER_ID;
                    break;
                }

    Sort((Entity**&)m_storage-
>m_interestList, size,
(OrderMode)order, sortMode);
          }
          else if (command ==
FILTER)
          {
                std::cout <<
"Fields for filter\n";
                std::cout << "0.
ID\n";
                std::cout << "1.
Date\n";
                std::cout << "2.
Interest\n";
                std::cout << "3.
Owner ID\n";
                std::cout <<
"Choose field: ";
                unsigned short
field;
                std::cin >>
field;

    std::cin.ignore();
                std::cout <<
'\n';

                if (field == 0)
                {
                    std::cout <<
"Input part of ID: ";
                    Id id;
                    std::cin >>
id;

    std::cin.ignore();
    Filter((Entity**&)m_storage
->m_interestList, size, id,
FieldMode::ID);
                }
                if (field == 1)
                {
                    std::cout <<
"Input date (if you don't want
to filter by the field, input
0)\n";
                    std::cout <<
"Day: ";
                    unsigned
short day;
                    std::cin >>
day;
                    std::cout <<
"Month: ";
                    unsigned
short month;
                    std::cin >>
month;
                    std::cout <<
"Year: ";
                    unsigned
short year;
                    std::cin >>
year;

    std::cin.ignore();
                    Entity::Date
date{ day, month, year };

    Filter((Entity**&)m_storage
->m_interestList, size, date,
FieldMode::DATE);
                }
                if (field == 2)
                {
                    std::cout <<
"Input part of interest: ";
                    std::string
interest{};

    std::getline(std::cin,
interest);

    Filter((Entity**&)m_storage
->m_interestList, size,
interest, FieldMode::CONTENT);
                }
                if (field == 3)
                {
```

```
                    std::cout <<
"Input part of owner ID: ";
                    Id ownerId;
                    std::cin >>
ownerId;

     std::cin.ignore();

     Filter((Entity**&)m_storage
->m_interestList, size, ownerId,
FieldMode::OWNER_ID);
               }
               m_storage-
>SetInterestListSize(size);
          }

          if (command == ADD ||
command == REMOVE || command ==
EDIT)
          {
               m_storage-
>SaveInterestList();
          }

          system("cls");
     }

     return 0;
}
```

ReminderListItem.h

```
#ifndef REMINDER_LIST_ITEM_H
#define REMINDER_LIST_ITEM_H
#include "AbstractMenuItem.h"
#include "Reminder.h"
#include "Storage.h"

namespace KMK
{
     class ReminderListItem :
public MenuItem
     {
     public:

     ReminderListItem(std::strin
g itemName, Storage* storage);

          int Run();

     private:
          Storage* m_storage{};
     };
}
#endif // !REMINDER_LIST_ITEM_H
```

ReminderListItem.cpp

```
#include "ReminderListItem.h"
#include <iostream>
#include <fstream>
#include "Add.h"
#include <Windows.h>
#include "Remove.h"
#include "Edit.h"
#include "Sort.h"
#include "Filter.h"
#include <iomanip>

using namespace KMK;

ReminderListItem::ReminderListIt
em(std::string itemName,
Storage* storage) :
     MenuItem(itemName)
{
     m_storage = storage;
}

int ReminderListItem::Run()
{
     enum  Command
     {
          RESET,
          ADD,
          REMOVE,
          EDIT,
          SORT,
          FILTER,
          ID,
          EXIT
     };

     unsigned short command = 0;

     while (command != EXIT)
     {
          size_t size =
m_storage-
>GetReminderListSize();

          unsigned short
maximumReminderLength = 8;
          for (Iteration i{}; i
< size; ++i)
          {
               if (m_storage-
>m_reminderList[i]-
```

```
>GetContent().length() >
maximumReminderLength)
                {

     maximumReminderLength =
m_storage->m_reminderList[i]-
>GetContent().length();
                }
          }

          std::cout <<
std::setw((11 + 3 + 3 + 5 + 11 +
maximumReminderLength + 1 + 3 +
3 + 5 + 8 +
GetItemName().length()) / 2) <<
GetItemName() << "\n\n";
          std::cout <<
std::setw(11) << "ID" << "|";
          std::cout <<
std::setw(3) << "dd" << "|";
          std::cout <<
std::setw(3) << "mm" << "|";
          std::cout <<
std::setw(5) << "yyyy" << "|";
          std::cout <<
std::setw(11) << "Owner ID" <<
"|";
          std::cout <<
std::setw(maximumReminderLength
+ 1) << "Reminder" << ":";
          std::cout <<
std::setw(3) << "dd" << "|";
          std::cout <<
std::setw(3) << "mm" << "|";
          std::cout <<
std::setw(5) << "yyyy";
          std::cout << '\n';

          for (Iteration i{}; i
< size; ++i)
          {
               std::cout <<
std::setw(11) << m_storage-
>m_reminderList[i]->GetId() <<
"|";
               std::cout <<
std::setw(3) << m_storage-
>m_reminderList[i]-
>GetDate().day << "|";
               std::cout <<
std::setw(3) << m_storage-
>m_reminderList[i]-
>GetDate().month << "|";
               std::cout <<
std::setw(5) << m_storage-
>m_reminderList[i]-
>GetDate().year << "|";
               std::cout <<
std::setw(11) << m_storage-
>m_reminderList[i]->GetOwnerId()
<< "|";
               std::cout <<
std::setw(maximumReminderLength
+ 1) << m_storage-
>m_reminderList[i]->GetContent()
<< ":";
               std::cout <<
std::setw(3) << m_storage-
>m_reminderList[i]-
>GetReminderTime().day << "|";
               std::cout <<
std::setw(3) << m_storage-
>m_reminderList[i]-
>GetReminderTime().month << "|";
               std::cout <<
std::setw(5) << m_storage-
>m_reminderList[i]-
>GetReminderTime().year;
               std::cout <<
'\n';
          }

          std::cout << '\n';
          std::cout << RESET <<
". Reset list\n";
          std::cout << ADD << ".
Add new reminder\n";
          std::cout << REMOVE <<
". Delete reminder\n";
          std::cout << EDIT <<
". Edit reminder\n";
          std::cout << SORT <<
". Sort list\n";
          std::cout << FILTER <<
". Filter list\n";
          std::cout << ID << ".
Choose ID\n";
          std::cout << EXIT <<
". Exit\n";
          std::cout << "Input
command: ";
          std::cin >> command;
          std::cin.ignore();
          std::cout << '\n';

          if (command == RESET)
          {
               m_storage-
>LoadReminderList();
          }
```

```
        else if (command ==
ADD)
        {
            Id ownerId;
            std::cout <<
"Input owner ID: ";
            std::cin >>
ownerId;

    std::cin.ignore();

            std::string
reminder{};
            std::cout <<
"Input reminder: ";

    std::getline(std::cin,
reminder);

            unsigned short
day;
            std::cout <<
"Input reminder day: ";
            std::cin >> day;

            unsigned short
month;
            std::cout <<
"Input reminder month: ";
            std::cin >>
month;

            unsigned short
year;
            std::cout <<
"Input reminder year: ";
            std::cin >> year;

    std::cin.ignore();

            SYSTEMTIME
systemTime;

    GetLocalTime(&systemTime);
            Reminder*
newReminder = new
Reminder(m_storage-
>GetNextReminderId(), {
systemTime.wDay,
systemTime.wMonth,
systemTime.wYear }, reminder,
ownerId, {day, month, year});


    Add((Entity**&)m_storage-
>m_reminderList, size,
newReminder);
            m_storage-
>SetReminderListSize(size);
        }
        else if (command ==
REMOVE)
        {
            std::cout <<
"Input ID: ";
            Id id;
            std::cin >> id;

    std::cin.ignore();


    Remove((Entity**&)m_storage
->m_reminderList, size, id);
            m_storage-
>SetReminderListSize(size);
        }
        else if (command ==
EDIT)
        {
            std::cout <<
"Input ID: ";
            Id id;
            std::cin >> id;

    std::cin.ignore();
            std::cout <<
'\n';

            std::cout <<
"Fields to edit\n";
            std::cout << "0.
ID\n";
            std::cout << "1.
Date\n";
            std::cout << "2.
Owner ID\n";
            std::cout << "3.
Reminder\n";
            std::cout << "4.
Reminder time\n";
            std::cout <<
"Choose field: ";
            unsigned short
fieldToChange;
            std::cin >>
fieldToChange;

    std::cin.ignore();
            std::cout <<
'\n';
```

```
            if (fieldToChange
== 0)
            {
                std::cout <<
"Input new ID: ";
                Id* newId =
new Id{};
                std::cin >>
*newId;

    std::cin.ignore();

    Edit((Entity**&)m_storage-
>m_reminderList, size, id,
(void*)newId, FieldMode::ID);
                delete
newId;
            }
            if (fieldToChange
== 1)
            {
                std::cout <<
"Input new date\n";
                std::cout <<
"Day: ";
                unsigned
short day;
                std::cin >>
day;
                std::cout <<
"Month: ";
                unsigned
short month;
                std::cin >>
month;
                std::cout <<
"Year: ";
                unsigned
short year;
                std::cin >>
year;

    std::cin.ignore();

    Entity::Date* newDate = new
Entity::Date{ day, month, year
};

    Edit((Entity**&)m_storage-
>m_reminderList, size, id,
(void*)newDate,
FieldMode::DATE);
                delete
newDate;
            }
            if (fieldToChange
== 2)
            {
                std::cout <<
"Input new owner ID: ";
                Id*
newOwnerId = new Id{};
                std::cin >>
*newOwnerId;

    std::cin.ignore();

    Edit((Entity**&)m_storage-
>m_reminderList, size, id,
(void*)newOwnerId,
FieldMode::OWNER_ID);
                delete
newOwnerId;

            }
            if (fieldToChange
== 3)
            {
                std::cout <<
"Input new reminder: ";
                std::string*
reminder = new std::string{};

    std::getline(std::cin,
*reminder);

    Edit((Entity**&)m_storage-
>m_reminderList, size, id,
(void*)reminder,
FieldMode::CONTENT);
                delete
reminder;
            }
            if (fieldToChange
== 4)
            {
                std::cout <<
"Input new reminder time\n";
                std::cout <<
"Day: ";
                unsigned
short day;
                std::cin >>
day;
                std::cout <<
"Month: ";
                unsigned
short month;
```

```
                    std::cin >>
month;
                    std::cout <<
"Year: ";
                    unsigned
short year;
                    std::cin >>
year;

     std::cin.ignore();

     Entity::Date*
newReminderDate = new
Entity::Date{ day, month, year
};

     Edit((Entity**&)m_storage-
>m_reminderList, size, id,
(void*)newReminderDate,
FieldMode::REMINDER_TIME);
                    delete
newReminderDate;
               }
          }
          else if (command ==
SORT)
          {
               std::cout <<
"Orders for sort\n";
               std::cout << "0.
Descending\n";
               std::cout << "1.
Ascending\n";
               std::cout <<
"Choose order: ";
               unsigned short
order;
               std::cin >>
order;
               std::cout <<
'\n';

               std::cout <<
"Fields for sort\n";
               std::cout << "0.
ID\n";
               std::cout << "1.
Date\n";
               std::cout << "2.
Owner ID\n";
               std::cout << "3.
Reminder\n";
               std::cout << "4.
Reminder time\n";
               std::cout <<
"Choose field: ";
               unsigned short
field;
               std::cin >>
field;

     std::cin.ignore();
               FieldMode
sortMode = (FieldMode)-1;
               switch (field)
               {
               case 0:
                    sortMode =
FieldMode::ID;
                    break;
               case 1:
                    sortMode =
FieldMode::DATE;
                    break;
               case 2:
                    sortMode =
FieldMode::OWNER_ID;
                    break;
               case 3:
                    sortMode =
FieldMode::CONTENT;
                    break;
               case 4:
                    sortMode =
FieldMode::REMINDER_TIME;
                    break;
               }

     Sort((Entity**&)m_storage-
>m_reminderList, size,
(OrderMode)order, sortMode);
          }
          else if (command ==
FILTER)
          {
               std::cout <<
"Fields for filter\n";
               std::cout << "0.
ID\n";
               std::cout << "1.
Date\n";
               std::cout << "2.
Owner ID\n";
               std::cout << "3.
Reminder\n";
               std::cout << "4.
Reminder time\n";
               std::cout <<
"Choose field: ";
```

```
                unsigned short
field;
                std::cin >>
field;

     std::cin.ignore();
                std::cout <<
'\n';

                if (field == 0)
                {
                     std::cout <<
"Input part of ID: ";
                     Id id;
                     std::cin >>
id;

     std::cin.ignore();

     Filter((Entity**&)m_storage
->m_reminderList, size, id,
FieldMode::ID);
                }
                if (field == 1)
                {
                     std::cout <<
"Input date (if you don't want
to filter by the field, input
0)\n";
                     std::cout <<
"Day: ";
                     unsigned
short day;
                     std::cin >>
day;
                     std::cout <<
"Month: ";
                     unsigned
short month;
                     std::cin >>
month;
                     std::cout <<
"Year: ";
                     unsigned
short year;
                     std::cin >>
year;

     std::cin.ignore();
                     Entity::Date
date{ day, month, year };

     Filter((Entity**&)m_storage
->m_reminderList, size, date,
FieldMode::DATE);
                }
                if (field == 2)
                {
                     std::cout <<
"Input part of owner ID: ";
                     Id ownerId;
                     std::cin >>
ownerId;

     std::cin.ignore();

     Filter((Entity**&)m_storage
->m_reminderList, size, ownerId,
FieldMode::OWNER_ID);

                }
                if (field == 3)
                {
                     std::cout <<
"Input part of reminder: ";
                     std::string
reminder{};

     std::getline(std::cin,
reminder);

     Filter((Entity**&)m_storage
->m_reminderList, size,
reminder, FieldMode::CONTENT);
                }
                if (field == 4)
                {
                     std::cout <<
"Input reminder time (if you
don't want to filter by the
field, input 0)\n";
                     std::cout <<
"Day: ";
                     unsigned
short day;
                     std::cin >>
day;
                     std::cout <<
"Month: ";
                     unsigned
short month;
                     std::cin >>
month;
                     std::cout <<
"Year: ";
                     unsigned
short year;
                     std::cin >>
year;
```

```
     std::cin.ignore();
                    Entity::Date
date{ day, month, year };

     Filter((Entity**&)m_storage
->m_reminderList, size, date,
FieldMode::REMINDER_TIME);
               }
               m_storage-
>SetReminderListSize(size);
          }

          if (command == ADD ||
command == REMOVE || command ==
EDIT)
          {
               m_storage-
>SaveReminderList();
          }

          system("cls");
     }

     return 0;
}
```

ThemeListItem.h

```
#ifndef THEME_LIST_ITEM_H
#define THEME_LIST_ITEM_H
#include "AbstractMenuItem.h"
#include "Theme.h"
#include "Storage.h"

namespace KMK
{
     class ThemeListItem :
public MenuItem
     {
     public:

     ThemeListItem(std::string
itemName, Storage* storage);

          int Run();

     private:
          Storage* m_storage{};
     };
}

#endif // !THEME_LIST_ITEM_H
```

ThemeListItem.cpp

```
#include "ThemeListItem.h"
#include <iostream>
#include <fstream>
#include "Add.h"
#include <Windows.h>
#include "Remove.h"
#include "Edit.h"
#include "Sort.h"
#include "Filter.h"
#include <iomanip>

using namespace KMK;

ThemeListItem::ThemeListItem(std
::string itemName, Storage*
storage) :
     MenuItem(itemName)
{
     m_storage = storage;
}

int ThemeListItem::Run()
{
     enum  Command
     {
          RESET,
          ADD,
          REMOVE,
          EDIT,
          SORT,
          FILTER,
          ID,
          EXIT
     };

     unsigned short command = 0;

     while (command != EXIT)
     {
          size_t size =
m_storage->GetThemeListSize();

          unsigned short
maximumThemeLength = 5;
          for (Iteration i{}; i
< size; ++i)
          {
               if (m_storage-
>m_themeList[i]-
>GetContent().length() >
maximumThemeLength)
               {
```

```
     maximumThemeLength =
m_storage->m_themeList[i]-
>GetContent().length();
                }
          }

          std::cout <<
std::setw((11 + 3 + 3 + 5 +
maximumThemeLength + 1 + 11 + 5
+ GetItemName().length()) / 2)
<< GetItemName() << "\n\n";
          std::cout <<
std::setw(11) << "ID" << "|";
          std::cout <<
std::setw(3) << "dd" << "|";
          std::cout <<
std::setw(3) << "mm" << "|";
          std::cout <<
std::setw(5) << "yyyy" << "|";
          std::cout <<
std::setw(maximumThemeLength +
1) << "Theme" << "|";
          std::cout <<
std::setw(11) << "Owner ID";
          std::cout << '\n';

          for (Iteration i{}; i
< size; ++i)
          {
               std::cout <<
std::setw(11) << m_storage-
>m_themeList[i]->GetId() << "|";
               std::cout <<
std::setw(3) << m_storage-
>m_themeList[i]->GetDate().day
<< "|";
               std::cout <<
std::setw(3) << m_storage-
>m_themeList[i]->GetDate().month
<< "|";
               std::cout <<
std::setw(5) << m_storage-
>m_themeList[i]->GetDate().year
<< "|";
               std::cout <<
std::setw(maximumThemeLength +
1) << m_storage->m_themeList[i]-
>GetContent() << "|";
               std::cout <<
std::setw(11) << m_storage-
>m_themeList[i]->GetOwnerId();
               std::cout <<
'\n';
          }
          std::cout << '\n';
          std::cout << RESET <<
". Reset list\n";
          std::cout << ADD << ".
Add new theme\n";
          std::cout << REMOVE <<
". Delete theme\n";
          std::cout << EDIT <<
". Edit theme\n";
          std::cout << SORT <<
". Sort list\n";
          std::cout << FILTER <<
". Filter list\n";
          std::cout << ID << ".
Choose ID\n";
          std::cout << EXIT <<
". Exit\n";
          std::cout << "Input
command: ";
          std::cin >> command;
          std::cin.ignore();
          std::cout << '\n';

          if (command == RESET)
          {
               m_storage-
>LoadThemeList();
          }
          else if (command ==
ADD)
          {
               std::string
theme{};
               std::cout <<
"Input theme: ";

     std::getline(std::cin,
theme);

               Id ownerId;
               std::cout <<
"Input owner ID: ";
               std::cin >>
ownerId;

     std::cin.ignore();

               SYSTEMTIME
systemTime;

     GetLocalTime(&systemTime);
               Theme* newTheme =
new Theme(m_storage-
>GetNextThemeId(), {
```

```
systemTime.wDay,
systemTime.wMonth,
systemTime.wYear }, theme,
ownerId);


     Add((Entity**&)m_storage-
>m_themeList, size, newTheme);
               m_storage-
>SetThemeListSize(size);
          }
          else if (command ==
REMOVE)
          {
               std::cout <<
"Input ID: ";
               Id id;
               std::cin >> id;

     std::cin.ignore();


     Remove((Entity**&)m_storage
->m_themeList, size, id);
               m_storage-
>SetThemeListSize(size);
          }
          else if (command ==
EDIT)
          {
               std::cout <<
"Input ID: ";
               Id id;
               std::cin >> id;

     std::cin.ignore();
               std::cout <<
'\n';

               std::cout <<
"Fields to edit\n";
               std::cout << "0.
ID\n";
               std::cout << "1.
Date\n";
               std::cout << "2.
Theme\n";
               std::cout << "3.
Owner ID\n";
               std::cout <<
"Choose field: ";
               unsigned short
fieldToChange;
               std::cin >>
fieldToChange;
     std::cin.ignore();
               std::cout <<
'\n';

               if (fieldToChange
== 0)
               {
                    std::cout <<
"Input new ID: ";
                    Id* newId =
new Id{};
                    std::cin >>
*newId;

     std::cin.ignore();

     Edit((Entity**&)m_storage-
>m_themeList, size, id,
(void*)newId, FieldMode::ID);
                    delete
newId;
               }
               if (fieldToChange
== 1)
               {
                    std::cout <<
"Input new date\n";
                    std::cout <<
"Day: ";
                    unsigned
short day;
                    std::cin >>
day;
                    std::cout <<
"Month: ";
                    unsigned
short month;
                    std::cin >>
month;
                    std::cout <<
"Year: ";
                    unsigned
short year;
                    std::cin >>
year;

     std::cin.ignore();

     Entity::Date* newDate = new
Entity::Date{ day, month, year
};

     Edit((Entity**&)m_storage-
>m_themeList, size, id,
```

```
(void*)newDate,
FieldMode::DATE);
                    delete
newDate;
               }
               if (fieldToChange
== 2)
               {
                    std::cout <<
"Input new theme: ";
                    std::string*
theme = new std::string{};

     std::getline(std::cin,
*theme);

     Edit((Entity**&)m_storage-
>m_themeList, size, id,
(void*)theme,
FieldMode::CONTENT);
                    delete
theme;
               }
               if (fieldToChange
== 3)
               {
                    std::cout <<
"Input new owner ID: ";
                    Id*
newOwnerId = new Id{};
                    std::cin >>
*newOwnerId;

     std::cin.ignore();

     Edit((Entity**&)m_storage-
>m_themeList, size, id,
(void*)newOwnerId,
FieldMode::OWNER_ID);
                    delete
newOwnerId;
               }
          }
          else if (command ==
SORT)
          {
               std::cout <<
"Orders for sort\n";
               std::cout << "0.
Descending\n";
               std::cout << "1.
Ascending\n";
               std::cout <<
"Choose order: ";
               unsigned short
order;
               std::cin >>
order;
               std::cout <<
'\n';

               std::cout <<
"Fields for sort\n";
               std::cout << "0.
ID\n";
               std::cout << "1.
Date\n";
               std::cout << "2.
Theme\n";
               std::cout << "3.
Owner ID\n";
               std::cout <<
"Choose field: ";
               unsigned short
field;
               std::cin >>
field;

     std::cin.ignore();
               FieldMode
sortMode = (FieldMode)-1;
               switch (field)
               {
               case 0:
                    sortMode =
FieldMode::ID;
                    break;
               case 1:
                    sortMode =
FieldMode::DATE;
                    break;
               case 2:
                    sortMode =
FieldMode::CONTENT;
                    break;
               case 3:
                    sortMode =
FieldMode::OWNER_ID;
                    break;
               }

     Sort((Entity**&)m_storage-
>m_themeList, size,
(OrderMode)order, sortMode);
          }
          else if (command ==
FILTER)
          {
```

```
                std::cout <<
"Fields for filter\n";
                std::cout << "0.
ID\n";
                std::cout << "1.
Date\n";
                std::cout << "2.
Theme\n";
                std::cout << "3.
Owner ID\n";
                std::cout <<
"Choose field: ";
                unsigned short
field;
                std::cin >>
field;

     std::cin.ignore();
                std::cout <<
'\n';

                if (field == 0)
                {
                     std::cout <<
"Input part of ID: ";
                     Id id;
                     std::cin >>
id;

     std::cin.ignore();

     Filter((Entity**&)m_storage
->m_themeList, size, id,
FieldMode::ID);
                }
                if (field == 1)
                {
                     std::cout <<
"Input date (if you don't want
to filter by the field, input
0)\n";
                     std::cout <<
"Day: ";
                     unsigned
short day;
                     std::cin >>
day;
                     std::cout <<
"Month: ";
                     unsigned
short month;
                     std::cin >>
month;
                     std::cout <<
"Year: ";
                     unsigned
short year;
                     std::cin >>
year;

     std::cin.ignore();
                     Entity::Date
date { day, month, year };

     Filter((Entity**&)m_storage
->m_themeList, size, date,
FieldMode::DATE);
                }
                if (field == 2)
                {
                     std::cout <<
"Input part of theme: ";
                     std::string
theme{};

     std::getline(std::cin,
theme);

     Filter((Entity**&)m_storage
->m_themeList, size, theme,
FieldMode::CONTENT);
                }
                if (field == 3)
                {
                     std::cout <<
"Input part of owner ID: ";
                     Id ownerId;
                     std::cin >>
ownerId;

     std::cin.ignore();

     Filter((Entity**&)m_storage
->m_themeList, size, ownerId,
FieldMode::OWNER_ID);
                }
                m_storage-
>SetThemeListSize(size);
          }

          if (command == ADD ||
command == REMOVE || command ==
EDIT)
          {
               m_storage-
>SaveThemeList();
          }

          system("cls");
     }
```

```
     return 0;
}
```

UserListItem.h

```
#ifndef USER_LIST_ITEM_H
#define USER_LIST_ITEM_H
#include "AbstractMenuItem.h"
#include "User.h"
#include "Storage.h"

namespace KMK
{
     class UserListItem : public
MenuItem
     {
     public:

     UserListItem(std::string
itemName, Storage* storage);

          int Run();

     private:
          Storage* m_storage{};
     };
}

#endif // !USER_LIST_ITEM_H
```

UserListItem.cpp

```
#include "UserListItem.h"
#include <iostream>
#include <fstream>
#include "Add.h"
#include <Windows.h>
#include "Remove.h"
#include "Edit.h"
#include "Sort.h"
#include "Filter.h"
#include <iomanip>

using namespace KMK;

UserListItem::UserListItem(std::
string itemName, Storage*
storage) :
     MenuItem(itemName)
{
     m_storage = storage;
}

int UserListItem::Run()
{
     enum  Command
     {
          RESET,
          ADD,
          REMOVE,
          EDIT,
          SORT,
          FILTER,
          ID,
          EXIT
     };

     unsigned short command = 0;

     while (command != EXIT)
     {
          unsigned short
maximumNameLength = 4;
          unsigned short
maximumLoginLength = 5;
          unsigned short
maximumPasswordLength = 8;
          for (Iteration i{}; i
< m_storage->GetUserListSize();
++i)
          {
               if (m_storage-
>m_userList[i]-
>GetName().length() >
maximumNameLength)
               {

     maximumNameLength =
m_storage->m_userList[i]-
>GetName().length();
               }
               if (m_storage-
>m_userList[i]-
>GetLogin().length() >
maximumLoginLength)
               {

     maximumLoginLength =
m_storage->m_userList[i]-
>GetLogin().length();
               }
               if (m_storage-
>m_userList[i]-
>GetPassword().length() >
maximumPasswordLength)
               {

     maximumPasswordLength =
```

```
m_storage->m_userList[i]-
>GetPassword().length();
                }
          }

          std::cout <<
std::setw((11 + 3 + 3 + 5 +
maximumNameLength + 1 +
maximumLoginLength + 1 +
maximumPasswordLength + 1 + 6 +
GetItemName().length()) / 2) <<
GetItemName() << "\n\n";
          std::cout <<
std::setw(11) << "ID" << "|";
          std::cout <<
std::setw(3) << "dd" << "|";
          std::cout <<
std::setw(3) << "mm" << "|";
          std::cout <<
std::setw(5) << "yyyy" << "|";
          std::cout <<
std::setw(maximumNameLength + 1)
<< "Name" << "|";
          std::cout <<
std::setw(maximumLoginLength +
1) << "Login" << "|";
          std::cout <<
std::setw(maximumPasswordLength
+ 1) << "Password";
          std::cout << '\n';

          for (Iteration i{}; i
< m_storage->GetUserListSize();
++i)
          {
               std::cout <<
std::setw(11) << m_storage-
>m_userList[i]->GetId() << "|";
               std::cout <<
std::setw(3) << m_storage-
>m_userList[i]->GetDate().day <<
"|";
               std::cout <<
std::setw(3) << m_storage-
>m_userList[i]->GetDate().month
<< "|";
               std::cout <<
std::setw(5) << m_storage-
>m_userList[i]->GetDate().year
<< "|";
               std::cout <<
std::setw(maximumNameLength + 1)
<< m_storage->m_userList[i]-
>GetName() << "|";
               std::cout <<
std::setw(maximumLoginLength +
1) << m_storage->m_userList[i]-
>GetLogin() << "|";
               std::cout <<
std::setw(maximumPasswordLength
+ 1) << m_storage-
>m_userList[i]->GetPassword();
               std::cout <<
'\n';
          }

          std::cout << '\n';
          std::cout << RESET <<
". Reset list\n";
          std::cout << ADD << ".
Add new user\n";
          std::cout << REMOVE <<
". Delete user\n";
          std::cout << EDIT <<
". Edit user\n";
          std::cout << SORT <<
". Sort list\n";
          std::cout << FILTER <<
". Filter list\n";
          std::cout << ID << ".
Choose ID\n";
          std::cout << EXIT <<
". Exit\n";
          std::cout << "Input
command: ";
          std::cin >> command;
          std::cin.ignore();
          std::cout << '\n';

          size_t size =
m_storage->GetUserListSize();

          if (command == RESET)
          {
               m_storage-
>LoadUserList();
          }
          else if (command ==
ADD)
          {
               std::string
name{};
               std::cout <<
"Input name: ";

     std::getline(std::cin,
name);
```

```
                std::string
login{};
                std::cout <<
"Input login: ";

     std::getline(std::cin,
login);

                std::string
password{};
                std::cout <<
"Input password: ";

     std::getline(std::cin,
password);

                SYSTEMTIME
systemTime;

     GetLocalTime(&systemTime);
                User* newUser =
new User(m_storage-
>GetNextUserId(), {
systemTime.wDay,
systemTime.wMonth,
systemTime.wYear }, name, login,
password);



     Add((Entity**&)m_storage-
>m_userList, size, newUser);
                m_storage-
>SetUserListSize(size);
          }
          else if (command ==
REMOVE)
          {
                std::cout <<
"Input ID: ";
                Id id;
                std::cin >> id;

     std::cin.ignore();



     Remove((Entity**&)m_storage
->m_userList, size, id);
                m_storage-
>SetUserListSize(size);
          }
          else if (command ==
EDIT)
          {
                std::cout <<
"Input ID: ";
                Id id;
                std::cin >> id;

     std::cin.ignore();
                std::cout <<
'\n';

                std::cout <<
"Fields to edit\n";
                std::cout << "0.
ID\n";
                std::cout << "1.
Date\n";
                std::cout << "2.
Name\n";
                std::cout << "3.
Login\n";
                std::cout << "4.
Password\n";
                std::cout <<
"Choose field: ";
                unsigned short
fieldToChange;
                std::cin >>
fieldToChange;

     std::cin.ignore();
                std::cout <<
'\n';

                if (fieldToChange
== 0)
                {
                     std::cout <<
"Input new ID: ";
                     Id* newId =
new Id{};
                     std::cin >>
*newId;

     std::cin.ignore();

     Edit((Entity**&)m_storage-
>m_userList, size, id,
(void*)newId, FieldMode::ID);
                     delete
newId;
                }
                if (fieldToChange
== 1)
                {
                     std::cout <<
"Input new date\n";
                     std::cout <<
"Day: ";
```

```
                    unsigned
short day;
                    std::cin >>
day;
                    std::cout <<
"Month: ";
                    unsigned
short month;
                    std::cin >>
month;
                    std::cout <<
"Year: ";
                    unsigned
short year;
                    std::cin >>
year;

     std::cin.ignore();

     Entity::Date* newDate = new
Entity::Date{ day, month, year
};

     Edit((Entity**&)m_storage-
>m_userList, size, id,
(void*)newDate,
FieldMode::DATE);
                    delete
newDate;
               }
               if (fieldToChange
== 2)
               {
                    std::cout <<
"Input new name: ";
                    std::string*
name = new std::string{};

     std::getline(std::cin,
*name);

     Edit((Entity**&)m_storage-
>m_userList, size, id,
(void*)name, FieldMode::NAME);
                    delete name;
               }
               if (fieldToChange
== 3)
               {
                    std::cout <<
"Input new login: ";
                    std::string*
login = new std::string{};

     std::getline(std::cin,
*login);

     Edit((Entity**&)m_storage-
>m_userList, size, id,
(void*)login, FieldMode::LOGIN);
                    delete
login;
               }
               if (fieldToChange
== 4)
               {
                    std::cout <<
"Input new password: ";
                    std::string*
password = new std::string{};

     std::getline(std::cin,
*password);

     Edit((Entity**&)m_storage-
>m_userList, size, id,
(void*)password,
FieldMode::PASSWORD);
                    delete
password;
               }
          }
          else if (command ==
SORT)
          {
               std::cout <<
"Orders for sort\n";
               std::cout << "0.
Descending\n";
               std::cout << "1.
Ascending\n";
               std::cout <<
"Choose order: ";
               unsigned short
order;
               std::cin >>
order;
               std::cout <<
'\n';

               std::cout <<
"Fields for sort\n";
               std::cout << "0.
ID\n";
               std::cout << "1.
Date\n";
               std::cout << "2.
Name\n";
```

```
                std::cout << "3.
Login\n";
                std::cout << "4.
Password\n";
                std::cout <<
"Choose field: ";
                unsigned short
field;
                std::cin >>
field;

     std::cin.ignore();
                FieldMode
sortMode = (FieldMode)-1;
                switch (field)
                {
                case 0:
                     sortMode =
FieldMode::ID;
                     break;
                case 1:
                     sortMode =
FieldMode::DATE;
                     break;
                case 2:
                     sortMode =
FieldMode::NAME;
                     break;
                case 3:
                     sortMode =
FieldMode::LOGIN;
                     break;
                case 4:
                     sortMode =
FieldMode::PASSWORD;
                     break;
                }

     Sort((Entity**&)m_storage-
>m_userList, size,
(OrderMode)order, sortMode);
          }
          else if (command ==
FILTER)
          {
                std::cout <<
"Fields for filter\n";
                std::cout << "0.
ID\n";
                std::cout << "1.
Date\n";
                std::cout << "2.
Name\n";
                std::cout << "3.
Login\n";
                std::cout << "4.
Password\n";
                std::cout <<
"Choose field: ";
                unsigned short
field;
                std::cin >>
field;

     std::cin.ignore();
                std::cout <<
'\n';

                if (field == 0)
                {
                     std::cout <<
"Input part of ID: ";
                     Id id;
                     std::cin >>
id;

     std::cin.ignore();

     Filter((Entity**&)m_storage
->m_userList, size, id,
FieldMode::ID);
                }
                if (field == 1)
                {
                     std::cout <<
"Input date (if you don't want
to filter by the field, input
0)\n";
                     std::cout <<
"Day: ";
                     unsigned
short day;
                     std::cin >>
day;
                     std::cout <<
"Month: ";
                     unsigned
short month;
                     std::cin >>
month;
                     std::cout <<
"Year: ";
                     unsigned
short year;
                     std::cin >>
year;

     std::cin.ignore();
                     Entity::Date
date{ day, month, year };
```

```
     Filter((Entity**&)m_storage
->m_userList, size, date,
FieldMode::DATE);
                }
                if (field == 2)
                {
                     std::cout <<
"Input part of name: ";
                     std::string
name{};

     std::getline(std::cin,
name);

     Filter((Entity**&)m_storage
->m_userList, size, name,
FieldMode::NAME);
                }
                if (field == 3)
                {
                     std::cout <<
"Input part of login: ";
                     std::string
login{};

     std::getline(std::cin,
login);

     Filter((Entity**&)m_storage
->m_userList, size, login,
FieldMode::LOGIN);
                }
                if (field == 4)
                {
                     std::cout <<
"Input part of password: ";
                     std::string
password{};

     std::getline(std::cin,
password);

     Filter((Entity**&)m_storage
->m_userList, size, password,
FieldMode::PASSWORD);
                }
                m_storage-
>SetUserListSize(size);
          }

          if (command == ADD ||
command == REMOVE || command ==
EDIT)
          {
                m_storage-
>SaveUserList();
          }

          system("cls");
     }

     return 0;
}
```

Storage.h

```
#ifndef STORAGE_H
#define STORAGE_H
#include "Dialogue.h"
#include "Interest.h"
#include "Reminder.h"
#include "Theme.h"
#include "User.h"
#include "IdCounter.h"

namespace KMK
{
     class Storage
     {
     public:
          Storage();
          Storage(std::string
dialogueListFilePath,
std::string
interestListFilePath,
               std::string
reminderListFilePath,
std::string themeListFilePath,
std::string userListFilePath,
               std::string
dialogueIdCounterFilePath,
std::string
interestIdCounterFilePath,
               std::string
reminderIdCounterFilePath,
std::string
themeIdCounterFilePath,
               std::string
userIdCounterFilePath);

          size_t
GetDialogueListSize();
          size_t
GetInterestListSize();
          size_t
GetReminderListSize();
          size_t
GetThemeListSize();
```

```
        size_t
GetUserListSize();

        void
SetDialogueListSize(size_t
size);
        void
SetInterestListSize(size_t
size);
        void
SetReminderListSize(size_t
size);
        void
SetThemeListSize(size_t size);
        void
SetUserListSize(size_t size);

        void
SaveDialogueList();
        void
SaveInterestList();
        void
SaveReminderList();
        void SaveThemeList();
        void SaveUserList();

        void
LoadDialogueList();
        void
LoadInterestList();
        void
LoadReminderList();
        void LoadThemeList();
        void LoadUserList();

        Id
GetNextDialogueId();
        Id
GetNextInterestId();
        Id
GetNextReminderId();
        Id GetNextThemeId();
        Id GetNextUserId();

        Dialogue**
m_dialogueList{};
        Interest**
m_interestList{};
        Reminder**
m_reminderList{};
        Theme** m_themeList{};
        User** m_userList{};

    private:
        std::string
m_dialogueListFilePath{};
        std::string
m_interestListFilePath{};
        std::string
m_reminderListFilePath{};
        std::string
m_themeListFilePath{};
        std::string
m_userListFilePath{};

        size_t
m_dialogueListSize{};
        size_t
m_interestListSize{};
        size_t
m_reminderListSize{};
        size_t
m_themeListSize{};
        size_t
m_userListSize{};

        std::string
m_dialogueIdCounterFilePath{};
        std::string
m_interestIdCounterFilePath{};
        std::string
m_reminderIdCounterFilePath{};
        std::string
m_themeIdCounterFilePath{};
        std::string
m_userIdCounterFilePath{};

        IdCounter
m_dialogueIdCounter{};
        IdCounter
m_interestIdCounter{};
        IdCounter
m_reminderIdCounter{};
        IdCounter
m_themeIdCounter{};
        IdCounter
m_userIdCounter{};
    };
}

#endif // !STORAGE_H
```

Storage.cpp

```
#include "Storage.h"
#include <fstream>

namespace KMK
{
```

```
Storage::Storage()
{
    m_dialogueList = {};
    m_interestList = {};
    m_reminderList = {};
    m_themeList = {};
    m_userList = {};

    m_dialogueListFilePath
= {};
    m_interestListFilePath
= {};
    m_reminderListFilePath
= {};
    m_themeListFilePath =
{};
    m_userListFilePath =
{};

    m_dialogueListSize =
0;
    m_interestListSize =
0;
    m_reminderListSize =
0;
    m_themeListSize = 0;
    m_userListSize = 0;


m_dialogueIdCounterFilePath
= {};

m_interestIdCounterFilePath
= {};

m_reminderIdCounterFilePath
= {};

m_themeIdCounterFilePath =
{};

m_userIdCounterFilePath =
{};

    m_dialogueIdCounter =
{};
    m_interestIdCounter=
{};
    m_reminderIdCounter =
{};
    m_themeIdCounter = {};
    m_userIdCounter = {};

}
```

```
Storage::Storage(std::strin
g dialogueListFilePath,
std::string
interestListFilePath,
          std::string
reminderListFilePath,
std::string themeListFilePath,
std::string userListFilePath,
          std::string
dialogueIdCounterFilePath,
std::string
interestIdCounterFilePath,
          std::string
reminderIdCounterFilePath,
std::string
themeIdCounterFilePath,
          std::string
userIdCounterFilePath)
     {
          m_dialogueListFilePath
= dialogueListFilePath;
          m_interestListFilePath
= interestListFilePath;
          m_reminderListFilePath
= reminderListFilePath;
          m_themeListFilePath =
themeListFilePath;
          m_userListFilePath =
userListFilePath;

          LoadDialogueList();
          LoadInterestList();
          LoadReminderList();
          LoadThemeList();
          LoadUserList();


     m_dialogueIdCounterFilePath
= dialogueIdCounterFilePath;

     m_interestIdCounterFilePath
= interestIdCounterFilePath;

     m_reminderIdCounterFilePath
= reminderIdCounterFilePath;

     m_themeIdCounterFilePath =
themeIdCounterFilePath;

     m_userIdCounterFilePath =
userIdCounterFilePath;


     m_dialogueIdCounter.SetIdCo
```

```
unter(dialogueIdCounterFilePath)
;

     m_interestIdCounter.SetIdCo
unter(interestIdCounterFilePath)
;

     m_reminderIdCounter.SetIdCo
unter(reminderIdCounterFilePath)
;

     m_themeIdCounter.SetIdCount
er(themeIdCounterFilePath);

     m_userIdCounter.SetIdCounte
r(userIdCounterFilePath);
     }

     size_t
Storage::GetDialogueListSize() {
return m_dialogueListSize; }
     size_t
Storage::GetInterestListSize() {
return m_interestListSize; }
     size_t
Storage::GetReminderListSize() {
return m_reminderListSize; }
     size_t
Storage::GetThemeListSize() {
return m_themeListSize; }
     size_t
Storage::GetUserListSize() {
return m_userListSize; }

     void
Storage::SetDialogueListSize(siz
e_t size) { m_dialogueListSize =
size; }
     void
Storage::SetInterestListSize(siz
e_t size) { m_interestListSize =
size; }
     void
Storage::SetReminderListSize(siz
e_t size) { m_reminderListSize =
size; }
     void
Storage::SetThemeListSize(size_t
size) { m_themeListSize = size;
}
     void
Storage::SetUserListSize(size_t
size) { m_userListSize = size; }

     void
Storage::SaveDialogueList()
     {
          std::ofstream
fileWrite(m_dialogueListFilePath
, std::ios::binary);

     fileWrite.write((char*)&m_d
ialogueListSize,
sizeof(size_t));
          for (Iteration i{}; i
< m_dialogueListSize; ++i)
          {
               Id id =
m_dialogueList[i]->GetId();

     fileWrite.write((char*)&id,
sizeof(Id));


     fileWrite.write((char*)&m_d
ialogueList[i]->GetDate(),
sizeof(Entity::Date));

               size_t stringSize
= m_dialogueList[i]-
>GetContent().length() + 1;

     fileWrite.write((char*)&str
ingSize, sizeof(size_t));

     fileWrite.write(m_dialogueL
ist[i]->GetContent().c_str(),
stringSize);

               id =
m_dialogueList[i]->GetOwnerId();

     fileWrite.write((char*)&id,
sizeof(Id));

               id =
m_dialogueList[i]-
>GetAdresseeId();

     fileWrite.write((char*)&id,
sizeof(Id));
          }
          fileWrite.close();
     }

     void
Storage::SaveInterestList()
     {
```

```
          std::ofstream
fileWrite(m_interestListFilePath
, std::ios::binary);

     fileWrite.write((char*)&m_i
nterestListSize,
sizeof(size_t));
          for (Iteration i{}; i
< m_interestListSize; ++i)
          {
               Id id =
m_interestList[i]->GetId();

     fileWrite.write((char*)&id,
sizeof(Id));



     fileWrite.write((char*)&m_i
nterestList[i]->GetDate(),
sizeof(Entity::Date));

               size_t stringSize
= m_interestList[i]-
>GetContent().length() + 1;

     fileWrite.write((char*)&str
ingSize, sizeof(size_t));

     fileWrite.write(m_interestL
ist[i]->GetContent().c_str(),
stringSize);

               id =
m_interestList[i]->GetOwnerId();

     fileWrite.write((char*)&id,
sizeof(Id));
          }
          fileWrite.close();
     }

     void
Storage::SaveReminderList()
     {
          std::ofstream
fileWrite(m_reminderListFilePath
, std::ios::binary);

     fileWrite.write((char*)&m_r
eminderListSize,
sizeof(size_t));
          for (Iteration i{}; i
< m_reminderListSize; ++i)
          {
               Id id =
m_reminderList[i]->GetId();

     fileWrite.write((char*)&id,
sizeof(Id));



     fileWrite.write((char*)&m_r
eminderList[i]->GetDate(),
sizeof(Entity::Date));

               size_t stringSize
= m_reminderList[i]-
>GetContent().length() + 1;

     fileWrite.write((char*)&str
ingSize, sizeof(size_t));

     fileWrite.write(m_reminderL
ist[i]->GetContent().c_str(),
stringSize);

               id =
m_reminderList[i]->GetOwnerId();

     fileWrite.write((char*)&id,
sizeof(Id));



     fileWrite.write((char*)&m_r
eminderList[i]-
>GetReminderTime(),
sizeof(Entity::Date));
          }
          fileWrite.close();
     }

     void
Storage::SaveThemeList()
     {
          std::ofstream
fileWrite(m_themeListFilePath,
std::ios::binary);

     fileWrite.write((char*)&m_t
hemeListSize, sizeof(size_t));
          for (Iteration i{}; i
< m_themeListSize; ++i)
          {
               Id id =
m_themeList[i]->GetId();

     fileWrite.write((char*)&id,
sizeof(Id));
```

```
     fileWrite.write((char*)&m_t
hemeList[i]->GetDate(),
sizeof(Entity::Date));

               size_t stringSize
= m_themeList[i]-
>GetContent().length() + 1;

     fileWrite.write((char*)&str
ingSize, sizeof(size_t));

     fileWrite.write(m_themeList
[i]->GetContent().c_str(),
stringSize);

               id =
m_themeList[i]->GetOwnerId();

     fileWrite.write((char*)&id,
sizeof(Id));
          }
          fileWrite.close();
     }

     void
Storage::SaveUserList()
     {
          std::ofstream
fileWrite(m_userListFilePath,
std::ios::binary);

     fileWrite.write((char*)&m_u
serListSize, sizeof(size_t));
          for (Iteration i{}; i
< m_userListSize; ++i)
          {
               Id id =
m_userList[i]->GetId();

     fileWrite.write((char*)&id,
sizeof(Id));



     fileWrite.write((char*)&m_u
serList[i]->GetDate(),
sizeof(Entity::Date));

               size_t stringSize
= m_userList[i]-
>GetName().length() + 1;

     fileWrite.write((char*)&str
ingSize, sizeof(size_t));

     fileWrite.write(m_userList[
i]->GetName().c_str(),
stringSize);

               stringSize =
m_userList[i]-
>GetLogin().length() + 1;

     fileWrite.write((char*)&str
ingSize, sizeof(size_t));

     fileWrite.write(m_userList[
i]->GetLogin().c_str(),
stringSize);

               stringSize =
m_userList[i]-
>GetPassword().length() + 1;

     fileWrite.write((char*)&str
ingSize, sizeof(size_t));

     fileWrite.write(m_userList[
i]->GetPassword().c_str(),
stringSize);
          }
          fileWrite.close();
     }

     void
Storage::LoadDialogueList()
     {
          std::ifstream
fileRead(m_dialogueListFilePath,
std::ios::binary);

     fileRead.read((char*)&m_dia
logueListSize, sizeof(size_t));
          m_dialogueList = new
Dialogue * [m_dialogueListSize]
{};
          for (Iteration i{}; i
< m_dialogueListSize; ++i)
          {
               Id id{};

     fileRead.read((char*)&id,
sizeof(Id));

               Entity::Date
date{};

     fileRead.read((char*)&date,
sizeof(Entity::Date));
```

```
                size_t
stringSize{};

     fileRead.read((char*)&strin
gSize, sizeof(size_t));
                char* message =
new char[stringSize] {};

     fileRead.read(message,
stringSize);

                Id ownerId{};

     fileRead.read((char*)&owner
Id, sizeof(Id));

                Id adresseeId{};

     fileRead.read((char*)&adres
seeId, sizeof(Id));

                m_dialogueList[i]
= new Dialogue{ id, date,
message, ownerId, adresseeId };

                delete[] message;
           }
           fileRead.close();
     }

     void
Storage::LoadInterestList()
     {
          std::ifstream
fileRead(m_interestListFilePath,
std::ios::binary);

     fileRead.read((char*)&m_int
erestListSize, sizeof(size_t));
          m_interestList = new
Interest * [m_interestListSize]
{};
          for (Iteration i{}; i
< m_interestListSize; ++i)
          {
                Id id{};

     fileRead.read((char*)&id,
sizeof(Id));

                Entity::Date
date{};

     fileRead.read((char*)&date,
sizeof(Entity::Date));

                size_t
stringSize{};

     fileRead.read((char*)&strin
gSize, sizeof(size_t));
                char* interest =
new char[stringSize] {};

     fileRead.read(interest,
stringSize);

                Id ownerId{};

     fileRead.read((char*)&owner
Id, sizeof(Id));

                m_interestList[i]
= new Interest{ id, date,
interest, ownerId };

                delete[]
interest;
           }
           fileRead.close();
     }

     void
Storage::LoadReminderList()
     {
          std::ifstream
fileRead(m_reminderListFilePath,
std::ios::binary);

     fileRead.read((char*)&m_rem
inderListSize, sizeof(size_t));
          m_reminderList = new
Reminder * [m_reminderListSize]
{};
          for (Iteration i{}; i
< m_reminderListSize; ++i)
          {
                Id id{};

     fileRead.read((char*)&id,
sizeof(Id));

                Entity::Date
date{};

     fileRead.read((char*)&date,
sizeof(Entity::Date));
```

```
                size_t
stringSize{};

     fileRead.read((char*)&strin
gSize, sizeof(size_t));
                char* reminder =
new char[stringSize] {};

     fileRead.read(reminder,
stringSize);

                Id ownerId{};

     fileRead.read((char*)&owner
Id, sizeof(Id));

                Entity::Date
reminderDate{};

     fileRead.read((char*)&remin
derDate, sizeof(Entity::Date));

                m_reminderList[i]
= new Reminder{ id, date,
reminder, ownerId, reminderDate
};

                delete[]
reminder;
          }
          fileRead.close();
     }

     void
Storage::LoadThemeList()
     {
          std::ifstream
fileRead(m_themeListFilePath,
std::ios::binary);

     fileRead.read((char*)&m_the
meListSize, sizeof(size_t));
          m_themeList = new
Theme * [m_themeListSize] {};
          for (Iteration i{}; i
< m_themeListSize; ++i)
          {
                Id id{};

     fileRead.read((char*)&id,
sizeof(Id));

                Entity::Date
date{};

     fileRead.read((char*)&date,
sizeof(Entity::Date));

                size_t
stringSize{};

     fileRead.read((char*)&strin
gSize, sizeof(size_t));
                char* theme = new
char[stringSize] {};

     fileRead.read(theme,
stringSize);

                Id ownerId{};

     fileRead.read((char*)&owner
Id, sizeof(Id));

                m_themeList[i] =
new Theme{ id, date, theme,
ownerId };

                delete[] theme;
          }
          fileRead.close();
     }

     void
Storage::LoadUserList()
     {
          std::ifstream
fileRead(m_userListFilePath,
std::ios::binary);

     fileRead.read((char*)&m_use
rListSize, sizeof(size_t));
          m_userList = new User
* [m_userListSize] {};
          for (Iteration i{}; i
< m_userListSize; ++i)
          {
                Id id{};

     fileRead.read((char*)&id,
sizeof(Id));

                Entity::Date
date{};

     fileRead.read((char*)&date,
sizeof(Entity::Date));
```

```
                size_t
stringSize{};

     fileRead.read((char*)&strin
gSize, sizeof(size_t));
                char* name = new
char[stringSize] {};

     fileRead.read(name,
stringSize);



     fileRead.read((char*)&strin
gSize, sizeof(size_t));
                char* login = new
char[stringSize] {};

     fileRead.read(login,
stringSize);



     fileRead.read((char*)&strin
gSize, sizeof(size_t));
                char* password =
new char[stringSize] {};

     fileRead.read(password,
stringSize);

                m_userList[i] =
new User{ id, date, name, login,
password };

                delete[] name;
                delete[] login;
                delete[]
password;
          }
          fileRead.close();
     }

     Id
Storage::GetNextDialogueId() {
return m_dialogueIdCounter(); }
     Id
Storage::GetNextInterestId() {
return m_interestIdCounter(); }
     Id
Storage::GetNextReminderId() {
return m_reminderIdCounter(); }
     Id
Storage::GetNextThemeId() {
return m_themeIdCounter(); }
     Id Storage::GetNextUserId()
{ return m_userIdCounter(); }
}
```

Add.h

```
#ifndef ADD_H
#define ADD_H
#include "Entity.h"

namespace KMK
{
     void Add(Entity**&
entities, size_t& size, Entity*
newElement);
}

#endif // !ADD_H
```

Add.cpp

```
#include "Add.h"
#include "Dialogue.h"
#include "Interest.h"
#include "Reminder.h"
#include "Theme.h"
#include "User.h"

using namespace KMK;

void KMK::Add(Entity**&
entities, size_t& size, Entity*
newElement)
{
     Entity** temp = new Entity
* [size + 1]{};

     for (Iteration i{}; i <
size; ++i)
     {
          if
(dynamic_cast<Dialogue*>(entitie
s[i]))
          {
               temp[i] = new
Dialogue{
dynamic_cast<Dialogue*>(entities
[i]) };
          }
          else if
(dynamic_cast<Interest*>(entitie
s[i]))
          {
               temp[i] = new
Interest{
```

```
dynamic_cast<Interest*>(entities
[i]) };
          }
          else if
(dynamic_cast<Reminder*>(entitie
s[i]))
          {
               temp[i] = new
Reminder{
dynamic_cast<Reminder*>(entities
[i]) };
          }
          else if
(dynamic_cast<Theme*>(entities[i
]))
          {
               temp[i] = new
Theme{
dynamic_cast<Theme*>(entities[i]
) };
          }
          else if
(dynamic_cast<User*>(entities[i]
))
          {
               temp[i] = new
User{
dynamic_cast<User*>(entities[i])
};
          }
     }

     temp[size] = newElement;

     for (Iteration i{}; i <
size; ++i)
     {
          delete entities[i];
     }
     delete[] entities;
     entities = temp;
     ++size;
}
```

Edit.h

```
#ifndef EDIT_H
#define EDIT_H
#include "GetField.h"

namespace KMK
{
     void Edit(Entity**&
entities, size_t size, Id
idToEdit, void* newField,
FieldMode mode);
}

#endif // !EDIT_H
```

Edit.cpp

```
#include "Edit.h"
#include "Dialogue.h"
#include "Interest.h"
#include "Reminder.h"
#include "Theme.h"
#include "User.h"

namespace KMK
{
     void Edit(Entity**&
entities, size_t size, Id
idToEdit, void* newField,
FieldMode mode)
     {
          unsigned short
themeNumber = 0;
          while (themeNumber <
size)
          {
               if
(entities[themeNumber]->GetId()
!= idToEdit)
               {

     ++themeNumber;
               }
               else
               {
                    break;
               }
          }

          if (themeNumber <
size)
          {
               if (mode ==
FieldMode::ID)
               {

     entities[themeNumber]-
>SetId(*(Id*)newField);
               }
               else if (mode ==
FieldMode::DATE)
               {

     entities[themeNumber]-
```

```
>SetDate(*(Entity::Date*)newFiel
d);
                }
                else if (mode ==
FieldMode::CONTENT)
                {

     dynamic_cast<TextEntity*>(e
ntities[themeNumber])-
>SetContent(*(std::string*)newFi
eld);
                }
                else if (mode ==
FieldMode::OWNER_ID)
                {

     dynamic_cast<TextEntity*>(e
ntities[themeNumber])-
>SetOwnerId(*(Id*)newField);
                }
                else if (mode ==
FieldMode::ADRESSEE_ID)
                {

     dynamic_cast<Dialogue*>(ent
ities[themeNumber])-
>SetAdresseeId(*(Id*)newField);
                }
                else if (mode ==
FieldMode::REMINDER_TIME)
                {

     dynamic_cast<Reminder*>(ent
ities[themeNumber])-
>SetReminderTime(*(Entity::Date*
)newField);
                }
                else if (mode ==
FieldMode::NAME)
                {

     dynamic_cast<User*>(entitie
s[themeNumber])-
>SetName(*(std::string*)newField
);
                }
                else if (mode ==
FieldMode::LOGIN)
                {

     dynamic_cast<User*>(entitie
s[themeNumber])-
>SetLogin(*(std::string*)newFiel
d);
                }
                else if (mode ==
FieldMode::PASSWORD)
                {

     dynamic_cast<User*>(entitie
s[themeNumber])-
>SetPassword(*(std::string*)newF
ield);
                }
          }
     }
}
```

Filter.h

```
#ifndef FILTER_H
#define FILTER_H
#include "GetField.h"

namespace KMK
{
     void Filter(Entity**&
entities, size_t& size, Id
fieldForSearch, FieldMode mode);
     void Filter(Entity**&
entities, size_t& size,
std::string fieldForSearch,
FieldMode mode);
     void Filter(Entity**&
entities, size_t& size,
Entity::Date fieldForSearch,
FieldMode mode);
}

#endif // !FILTER_H
```

Filter.cpp

```
#include "Filter.h"
#include "Constants.h"
#include <cmath>
#include "Dialogue.h"
#include "Interest.h"
#include "Reminder.h"
#include "Theme.h"
#include "User.h"

namespace KMK
{
     void
CreateNewList(Entity**&
entities, size_t& size, bool*
indexes, size_t newSize)
     {
```

```
          Entity**
filteredEntities = new Entity *
[newSize] {};
          Iteration
numberOfEntity = 0;
          for (Iteration i{}; i
< size; ++i)
          {
               if (indexes[i] ==
true)
               {
                    if
(dynamic_cast<Dialogue*>(entitie
s[i]))
                    {

     filteredEntities[numberOfEn
tity] = new Dialogue{
dynamic_cast<Dialogue*>(entities
[i]) };
                    }
                    else if
(dynamic_cast<Interest*>(entitie
s[i]))
                    {

     filteredEntities[numberOfEn
tity] = new Interest{
dynamic_cast<Interest*>(entities
[i]) };
                    }
                    else if
(dynamic_cast<Reminder*>(entitie
s[i]))
                    {

     filteredEntities[numberOfEn
tity] = new Reminder{
dynamic_cast<Reminder*>(entities
[i]) };
                    }
                    else if
(dynamic_cast<Theme*>(entities[i
]))
                    {

     filteredEntities[numberOfEn
tity] = new Theme{
dynamic_cast<Theme*>(entities[i]
) };
                    }
                    else if
(dynamic_cast<User*>(entities[i]
))
                    {

     filteredEntities[numberOfEn
tity] = new User{
dynamic_cast<User*>(entities[i])
};
                    }

     ++numberOfEntity;
               }
               delete
entities[i];
          }
          delete[] entities;
          entities =
filteredEntities;
          size = newSize;
     }

     void Filter(Entity**&
entities, size_t& size, Id
fieldForSearch, FieldMode mode)
     {
          unsigned long int tens
= 10;
          unsigned short
numberOfDigits = 1;
          while (fieldForSearch
/ tens != 0)
          {
               tens *= 10;
               ++numberOfDigits;
          }

          bool* indexes = new
bool[size] {};
          size_t newSize = 0;
          for (Iteration i{}; i
< MAXIMUM_NUMBER_OF_DIGITS_IN_ID
- numberOfDigits + 1; ++i)
          {
               for (Iteration
j{}; j < size; j++)
               {
                    if
(fieldForSearch ==
(GetIdField(entities[j], mode) /
(int)pow(10, i)) % tens)
                    {
                         if
(indexes[j] != true)
                         {

     indexes[j] = true;

     ++newSize;
```

```
                    }
                }
            }
        }


    CreateNewList(entities,
size, indexes, newSize);
    }

    void Filter(Entity**&
entities, size_t& size,
std::string fieldForSearch,
FieldMode mode)
    {
        unsigned short
fieldForSearchLength =
fieldForSearch.length();

        bool* indexes = new
bool[size] {};
        int newSize = 0;
        for (Iteration i{}; i
< size; ++i)
        {
            for (Iteration
j{}; j <
GetTextField(entities[i],
mode).length() -
fieldForSearchLength + 1; ++j)
            {
                std::string
temp{};
                for
(Iteration k{}; k <
fieldForSearchLength; ++k)
                {
                    temp +=
GetTextField(entities[i],
mode)[j + k];
                }
                if (temp ==
fieldForSearch)
                {

    indexes[i] = true;

    ++newSize;
                    break;
                }
            }
        }


    CreateNewList(entities,
size, indexes, newSize);
    }

    void Filter(Entity**&
entities, size_t& size,
Entity::Date fieldForSearch,
FieldMode mode)
    {
        bool* indexes = new
bool[size] {};
        unsigned short newSize
= 0;
        for (Iteration i{}; i
< size; ++i)
        {
            if
((GetDateField(entities[i],
mode).day == fieldForSearch.day
|| fieldForSearch.day == 0) &&

    (GetDateField(entities[i],
mode).month ==
fieldForSearch.month ||
fieldForSearch.month == 0) &&

    (GetDateField(entities[i],
mode).year ==
fieldForSearch.year ||
fieldForSearch.year == 0))
            {
                indexes[i] =
true;
                ++newSize;
            }
        }


    CreateNewList(entities,
size, indexes, newSize);
    }
}
```

Remove.h

```
#ifndef REMOVE_H
#define REMOVE_H
#include "Entity.h"

namespace KMK
{
    void Remove(Entity**&
entites, size_t& size, Id
idToRemove);
```

```
}

#endif // !REMOVE_H
```

Remove.cpp

```
#include "Remove.h"
#include "Dialogue.h"
#include "Interest.h"
#include "Reminder.h"
#include "Theme.h"
#include "User.h"

using namespace KMK;

void KMK::Remove(Entity**& entites, size_t& size, Id idToRemove)
{
     bool found = false;
     for (Iteration i{}; i < size; ++i)
     {
          if (idToRemove == entites[i]->GetId())
          {
               found = true;
               break;
          }
     }

     if (found == true)
     {
          Entity** temp = new Entity * [size - 1]{};
          unsigned short tempElementNumber = 0;
          for (Iteration i{}; i < size; ++i)
          {
               if (entites[i]->GetId() != idToRemove)
               {
                    if (dynamic_cast<Dialogue*>(entites[i]))
                    {

     temp[tempElementNumber] = new Dialogue{ dynamic_cast<Dialogue*>(entites[i]) };
                    }
                    else if (dynamic_cast<Interest*>(entites[i]))
                    {

     temp[tempElementNumber] = new Interest{ dynamic_cast<Interest*>(entites[i]) };
                    }
                    else if (dynamic_cast<Reminder*>(entites[i]))
                    {

     temp[tempElementNumber] = new Reminder{ dynamic_cast<Reminder*>(entites[i]) };
                    }
                    else if (dynamic_cast<Theme*>(entites[i]))
                    {

     temp[tempElementNumber] = new Theme{ dynamic_cast<Theme*>(entites[i]) };
                    }
                    else if (dynamic_cast<User*>(entites[i]))
                    {

     temp[tempElementNumber] = new User{ dynamic_cast<User*>(entites[i]) };
                    }

     ++tempElementNumber;
               }
          }

          for (Iteration i{}; i < size; ++i)
          {
               delete entites[i];
          }
          delete[] entites;
          entites = temp;
          --size;
     }
```

```
}
```

Sort.h

```
#ifndef SORT_H
#define SORT_H
#include "GetField.h"

namespace KMK
{
     enum class OrderMode
     {
          DESCENDING,
          ASCENDING
     };

     void Sort(Entity**& entities, size_t size, OrderMode order, FieldMode mode);
}

#endif // !SORT_H
```

Sort.cpp

```
#include "Sort.h"
#include <iostream>

namespace KMK
{
     void JustSwap(Entity*& firstEntity, Entity*& secondEntity)
     {
          Entity* temp = firstEntity;
          firstEntity = secondEntity;
          secondEntity = temp;
     }

     void Swap(Entity*& firstEntity, Entity*& secondEntity, OrderMode order, FieldMode mode)
     {
          if (mode == FieldMode::ID || mode == FieldMode::OWNER_ID || mode == FieldMode::ADRESSEE_ID)
          {
               if (order == OrderMode::DESCENDING && GetIdField(firstEntity, mode) < GetIdField(secondEntity, mode) ||
                    order == OrderMode::ASCENDING && GetIdField(firstEntity, mode) > GetIdField(secondEntity, mode))
               {

                    JustSwap(firstEntity, secondEntity);
               }
          }
          else if (mode == FieldMode::CONTENT || mode == FieldMode::NAME || mode == FieldMode::LOGIN || mode == FieldMode::PASSWORD)
          {
               Iteration letter = 0;
               while (GetTextField(firstEntity, mode)[letter] == GetTextField(secondEntity, mode)[letter] &&
                    letter < GetTextField(firstEntity, mode).length() - 1 && letter < GetTextField(secondEntity, mode).length() - 1)
               {
                    ++letter;
               }

               if (order == OrderMode::DESCENDING && GetTextField(firstEntity, mode)[letter] < GetTextField(secondEntity, mode)[letter] ||
                    order == OrderMode::ASCENDING && GetTextField(firstEntity, mode)[letter] > GetTextField(secondEntity, mode)[letter])
               {

                    JustSwap(firstEntity, secondEntity);
               }
          }
          else if (mode == FieldMode::DATE || mode == FieldMode::REMINDER_TIME)
```

```
          {
               if (order ==
OrderMode::DESCENDING &&

     (GetDateField(firstEntity,
mode).year <
GetDateField(secondEntity,
mode).year ||

     GetDateField(firstEntity,
mode).year ==
GetDateField(secondEntity,
mode).year &&

     (GetDateField(firstEntity,
mode).month <
GetDateField(secondEntity,
mode).month ||

     GetDateField(firstEntity,
mode).month ==
GetDateField(secondEntity,
mode).month &&

     GetDateField(firstEntity,
mode).day <
GetDateField(secondEntity,
mode).day)) ||
                    order ==
OrderMode::ASCENDING &&

     (GetDateField(firstEntity,
mode).year >
GetDateField(secondEntity,
mode).year ||

     GetDateField(firstEntity,
mode).year ==
GetDateField(secondEntity,
mode).year &&

     (GetDateField(firstEntity,
mode).month >
GetDateField(secondEntity,
mode).month ||

     GetDateField(firstEntity,
mode).month ==
GetDateField(secondEntity,
mode).month &&

     GetDateField(firstEntity,
mode).day >
GetDateField(secondEntity,
mode).day)))
               {

     JustSwap(firstEntity,
secondEntity);
               }
          }
     }

     void Sort(Entity**&
entities, size_t size, OrderMode
order, FieldMode mode)
     {
          if (size > 1)
          {
               short
bottomBorder = 0;
               short upperBorder
= size - 1;
               while
(upperBorder - bottomBorder > 0)
               {
                    for
(Iteration i = bottomBorder; i <
upperBorder; ++i)
                    {

     Swap(entities[i],
entities[i + 1], order, mode);
                    }
                    --
upperBorder;

                    for
(Iteration i = upperBorder; i >
bottomBorder; --i)
                    {

     Swap(entities[i - 1],
entities[i], order, mode);
                    }

     ++bottomBorder;
               }
          }
     }
}
```

GetField.h

```
#ifndef GET_FIELD_H
#define GET_FIELD_H
#include "Entity.h"
#include <string>

namespace KMK
```

```
{
     enum class FieldMode
     {
          ID,
          DATE,
          CONTENT,
          OWNER_ID,
          ADRESSEE_ID,
          REMINDER_TIME,
          NAME,
          LOGIN,
          PASSWORD
     };

     Id GetIdField(Entity*
entity, FieldMode mode);
     std::string
GetTextField(Entity* entity,
FieldMode mode);
     Entity::Date
GetDateField(Entity* entity,
FieldMode mode);
}

#endif // !GET_FIELD_H
```

GetField.cpp

```
#include "GetField.h"
#include "Dialogue.h"
#include "User.h"
#include "Reminder.h"

namespace KMK
{
     Id GetIdField(Entity*
entity, FieldMode mode)
     {
          if (mode ==
FieldMode::ID)
          {
               return entity-
>GetId();
          }
          else if (mode ==
FieldMode::OWNER_ID)
          {
               return
dynamic_cast<TextEntity*>(entity
)->GetOwnerId();
          }
          else if (mode ==
FieldMode::ADRESSEE_ID)
          {
               return
dynamic_cast<Dialogue*>(entity)-
>GetAdresseeId();
          }
     }

     std::string
GetTextField(Entity* entity,
FieldMode mode)
     {
          if (mode ==
FieldMode::CONTENT)
          {
               return
dynamic_cast<TextEntity*>(entity
)->GetContent();
          }
          else if (mode ==
FieldMode::NAME)
          {
               return
dynamic_cast<User*>(entity)-
>GetName();
          }
          else if (mode ==
FieldMode::LOGIN)
          {
               return
dynamic_cast<User*>(entity)-
>GetLogin();
          }
          else if (mode ==
FieldMode::PASSWORD)
          {
               return
dynamic_cast<User*>(entity)-
>GetPassword();
          }
     }

     Entity::Date
GetDateField(Entity* entity,
FieldMode mode)
     {
          if (mode ==
FieldMode::DATE)
          {
               return entity-
>GetDate();
          }
          else if (mode ==
FieldMode::REMINDER_TIME)
          {
```

```
                return
dynamic_cast<Reminder*>(entity)-
>GetReminderTime();
          }
     }
}
```

Main.cpp

```
#include "Menu.h"
#include "UserListItem.h"
#include "DialogueListItem.h"
#include "InterestListItem.h"
#include "ReminderListItem.h"
#include "ThemeListItem.h"
#include <iostream>
#include "Storage.h"

using namespace KMK;

int main()
{
     Storage* storage = new
Storage("Dialogue database.dat",
"Interst database.dat",
          "Reminder
database.dat", "Theme
database.dat", "User
database.dat", "Dialogue
IDs.dat", "Interst IDs.dat",
          "Reminder IDs.dat",
"Theme IDs.dat", "User
IDs.dat");

     UserListItem users =
UserListItem("User list",
storage);
     DialogueListItem dialogs =
DialogueListItem("Dialogue
list", storage);
     InterestListItem interests
= InterestListItem("Interest
list", storage);
     ReminderListItem reminders
= ReminderListItem("Reminder
list", storage);
     ThemeListItem themes =
ThemeListItem("Theme list",
storage);

     Menu menu = Menu("Chat
Bot", new MenuItem*[5] { &users,
&dialogs, &interests,
&reminders, &themes }, 5);
```

```
                std::cin >> menu;

                return 0;
          }
}
```

Демонстрация:

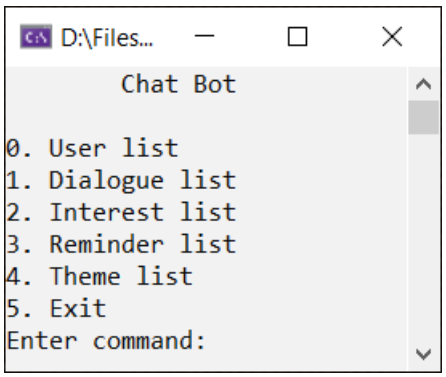

Рисунок 2. Главное меню

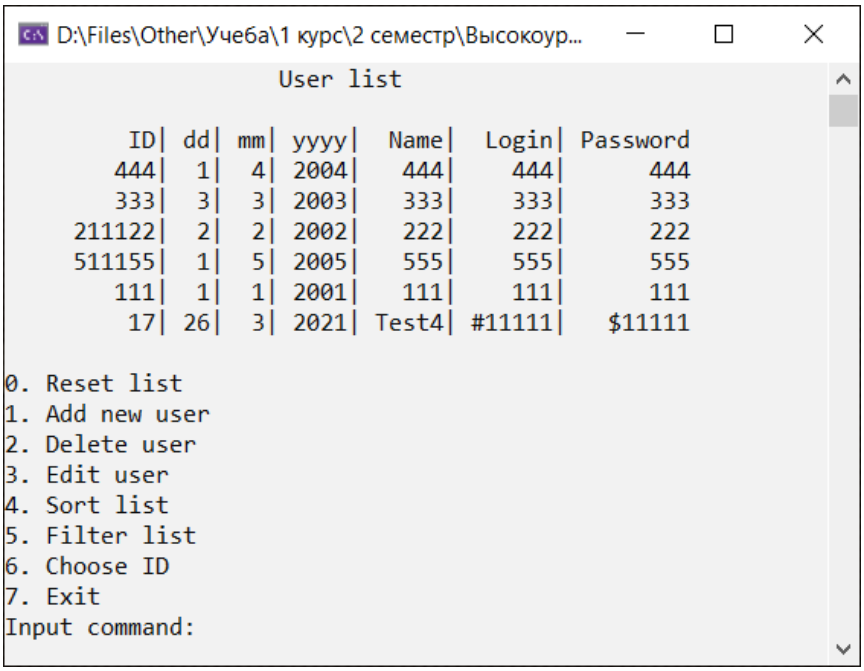

Рисунок 3. База данных пользователей

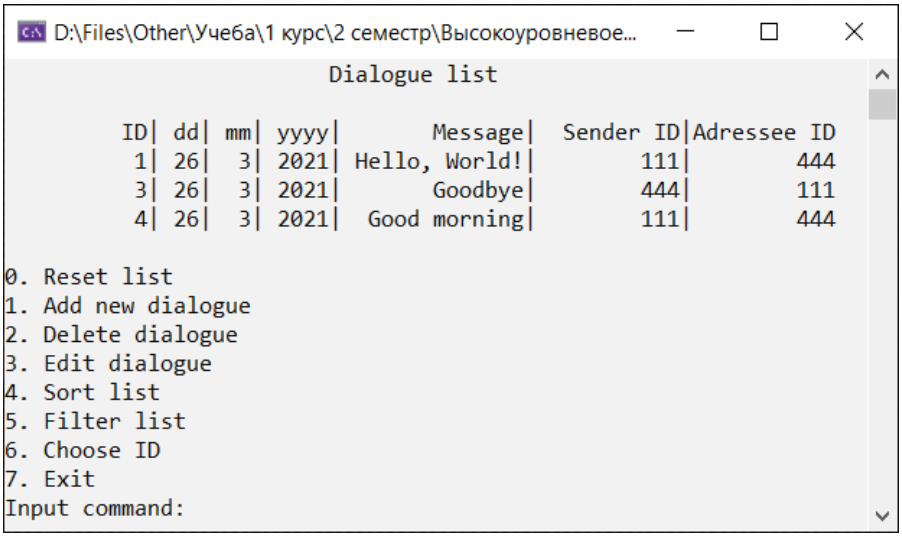

Рисунок 4. База данных диалогов

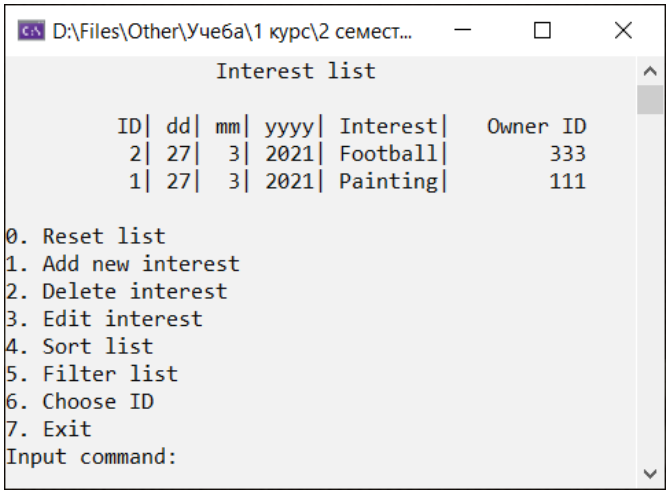

Рисунок 5. База данных интересов

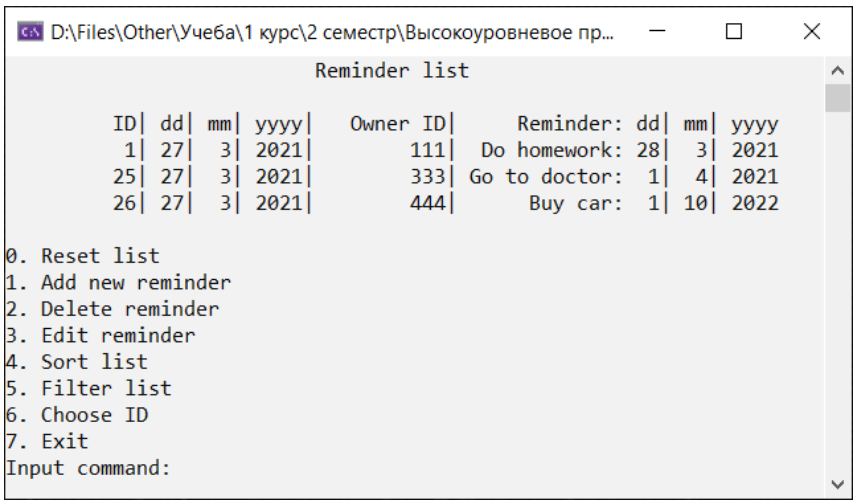

Рисунок 6. База данных напоминаний

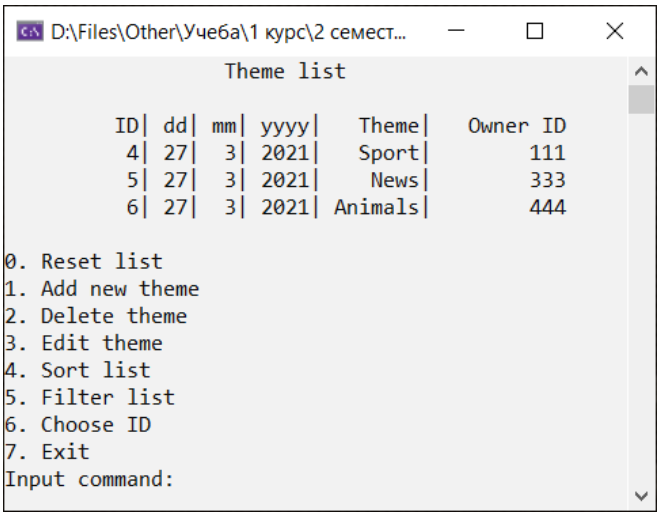

Рисунок 7. База данных тем

Вывод: в ходе выполнения лабораторной работы были получены практические навыки работы с dynamic_cast, контейнерами, string.