Выполнил

студент группы КТбо1-2 А. С. Мумладзе

Принял

доцент кафедры САиТ В. С. Лапшин

Таганрог 2024

Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное

учреждение высшего образования

«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

(ФГАОУ ВО «ЮФУ»)

Институт компьютерных технологий и информационной безопасности

Кафедра системного анализа и телекоммуникаций

ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ №2

по дисциплине «Основы алгоритмизации и программирования»

**«РАЗРАБОТКА ДИАГРАММЫ КЛАССОВ НА С++»**

Вариант 19

# Содержание

Содержание 2

Техническое задание 3

Цель задания 3

Задача 3

Ход работы 4

Структура UML-диаграммы 4

Написание программы 4

Вывод 5

Листинг 6

# Техническое задание

## Цель задания

Используя среду разработки MS Visual Studio, разработать и реализовать диаграмму классов на С++.

## Задача

Задачи индивидуального задания:

1. Разработать UML-диаграмму классов (в draw.io);
2. Реализовать структуру классов на С++ на основе созданной   
   UML-диаграммы;
3. Разработать методы контроля и редактирования параметров объектов.

Заданием моего варианта стала диаграмма классов для работы с журналом безопасности приложения пользователя. Критерием является наличие следующих классов: пользователя, приложения, журнала безопасности приложения.

# Ход работы

## Структура UML-диаграммы

Классы я реализовал следующим образом:

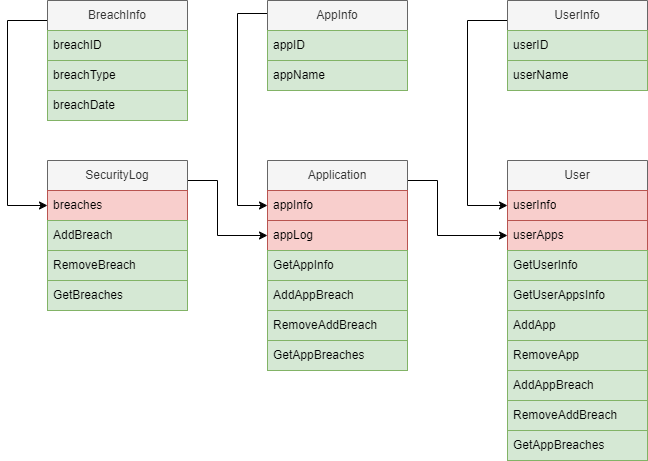


Рисунок 1 - UML-диаграмма классов

1. Класс SecurityLog. Содержит приватное поле, представляющее собой vector список нарушений, каждое из которых представляет собой структуру BreachInfo, содержащую информацию о нарушении. Методы этого класса:

* AddBreach – добавляет в лог новое нарушение.
* RemoveBreach – удаляет из лога нарушение по его ID.
* GetBreaches – возвращает вектор нарушений.

1. Класс Application. Содержит приватное поле appInfo, представляющее собой структуру AppInfo с содержащейся в ней информацией о приложении (ID и название приложения), а также объект класса SecurityLog. Методы:

* GetAppInfo – возвращает структуру AppInfo с данными приложения.
* AddAppBreach – добавляет в лог приложения новое нарушение
* RemoveAppBreach – удаляет из лога приложения нарушение по ID
* GetAppBreaches – возвращает вектор нарушений приложения.

1. Класс User. Содержит приватное поле userInfo – структуру UserInfo с данными о пользователе (ID и userName) – и вектор, содержащий объекты класса Application – список приложений пользователя. Методы:

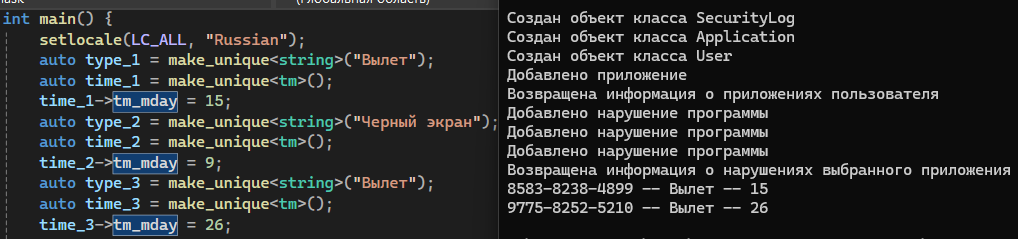
* GetUserInfo – возвращает UserInfo с данными о пользователе.
* GetUserAppsInfo – возвращает вектор с данными о приложениях пользователя (вектор AppInfo)
* AddApp – добавляет новое приложение в вектор приложений
* RemoveApp – удаляет приложение из вектора по ID приложения.
* AddAppBreach – добавляет в приложение с выбранным ID новое нарушение
* RemoveAppBreach – удаляет из выбранного по ID приложения нарушение по полученному ID нарушения.

## Написание программы

Особенностью реализации программы стало активное использование unique\_ptr. Это тип указателя из С++, который осуществляет уникальное хранение данных, без возможности копирования указателя, а также с автоматическим удалением данных (очистке памяти) при очистке указателя (вручную или при выходе из области видимости). Оно заменяет собой оперирование объектов через new/delete, что позволяет не следить более за утечкой памяти, взамен требуя грамотное оперирование через передачи указателя через функцию move().

В остальном реализация вышеуказанной диаграммы классов произведена стандартным для C++ способом: в Class.h прописано объявление классов, их полей и методов; в Class.cpp прописано определение всех методов классов; в main.cpp представлен пример создания объектов классов, а также вызова методов класса для оперирования данными классов.

## Пример программы



# Вывод

В ходе индивидуального задания я научился разрабатывать UML-диаграммы в графическом приложении draw.io, вновь ознакомился с методами реализации классов на С++, а также научился пользоваться unique\_ptr решением динамического выделения памяти языка С++.

# Листинг

## Class.h

1. #pragma once
2. #include <string>
3. #include <vector>
4. #include <memory>
5. #include <random>
6. #include <ctime>
7. #include <iostream>
8. using namespace std;
9. static unique\_ptr<string> MakeUUID();
10. struct BreachInfo {
11. unique\_ptr<string> breachID;
12. unique\_ptr<string> breachType;
13. unique\_ptr<tm> breachDate;
14. BreachInfo() = default;
15. BreachInfo(unique\_ptr<string> id, unique\_ptr<string> type, unique\_ptr<tm> date)
16. : breachID(move(id)), breachType(move(type)), breachDate(move(date)) {}
17. };
18. class SecurityLog {
19. private:
20. unique\_ptr<vector<unique\_ptr<BreachInfo>>> breaches;
21. public:
22. SecurityLog();
23. unique\_ptr<vector<unique\_ptr<BreachInfo>>> GetBreaches(unique\_ptr<string> breachType);
24. void AddBreach(unique\_ptr<string> breachType, unique\_ptr<tm> breachDate);
25. void RemoveBreach(unique\_ptr<string> breachID);
26. };
27. struct AppInfo {
28. unique\_ptr<string> appID;
29. unique\_ptr<string> appName;
30. AppInfo() = default;
31. AppInfo(unique\_ptr<string> id, unique\_ptr<string> name)
32. : appID(move(id)), appName(move(name)) {}
33. };
34. class Application {
35. private:
36. unique\_ptr<AppInfo> appInfo;
37. unique\_ptr<SecurityLog> appLog;
38. public:
39. Application(unique\_ptr<string> appName, unique\_ptr<SecurityLog> appLog);
40. unique\_ptr<AppInfo> GetAppInfo();
41. unique\_ptr<vector<unique\_ptr<BreachInfo>>> GetAppBreaches(unique\_ptr<string> breachType);
42. void AddAppBreach(unique\_ptr<string> breachType, unique\_ptr<tm> breachDate);
43. void RemoveAppBreach(unique\_ptr<string> breachID);
44. };
45. struct UserInfo {
46. unique\_ptr<string> userID;
47. unique\_ptr<string> userName;
48. UserInfo() = default;
49. UserInfo(unique\_ptr<string> id, unique\_ptr<string> name)
50. : userID(move(id)), userName(move(name)) {}
51. };
52. class User {
53. private:
54. unique\_ptr<UserInfo> userInfo;
55. unique\_ptr<vector<unique\_ptr<Application>>> userApps;
56. public:
57. User(unique\_ptr<string> userName);
58. unique\_ptr<UserInfo> GetUserInfo();
59. unique\_ptr<vector<unique\_ptr<AppInfo>>> GetUserAppsInfo();
60. void AddApp(unique\_ptr<Application> app);
61. void RemoveApp(unique\_ptr<string> appID);
62. void AddAppBreach(unique\_ptr<string> appID, unique\_ptr<string> breachType, unique\_ptr<tm> breachDate);
63. void RemoveAppBreach(unique\_ptr<string> appID, unique\_ptr<string> breachID);
64. unique\_ptr<vector<unique\_ptr<BreachInfo>>> GetAppBreaches(unique\_ptr<string> appID, unique\_ptr<string> breachType);
65. };

## Class.cpp

1. #include "Class.h"
2. static unique\_ptr<string> MakeUUID() {
3. random\_device rd;
4. mt19937 gen(rd());
5. uniform\_int\_distribution<> dis(0, 9999);
6. char id[16];
7. snprintf(id, sizeof(id), "%04d-%04d-%04d", dis(gen), dis(gen), dis(gen));
8. return make\_unique<string>(id);
9. }
10. //
11. // Класс SecurityLog
12. //
13. SecurityLog::SecurityLog() {
14. this->breaches = make\_unique<vector<unique\_ptr<BreachInfo>>>();
15. cout << "Создан объект класса SecurityLog\n";
16. }
17. unique\_ptr<vector<unique\_ptr<BreachInfo>>> SecurityLog::GetBreaches(unique\_ptr<string> breachType) {
18. bool isCopyAll = false;
19. if (breachType == nullptr)
20. isCopyAll = true;
21. auto breachesCopy = make\_unique<vector<unique\_ptr<BreachInfo>>>();
22. for (const auto& breach : \*breaches) {
23. if ((\*breach->breachType == \*breachType) || isCopyAll) {
24. auto info = make\_unique<BreachInfo>();
25. info->breachID = make\_unique<string>(\*breach->breachID);
26. info->breachType = make\_unique<string>(\*breach->breachType);
27. info->breachDate = make\_unique<tm>(\*breach->breachDate);
28. breachesCopy->push\_back(move(info));
29. }
30. }
31. return breachesCopy;
32. }
33. void SecurityLog::AddBreach(unique\_ptr<string> breachType, unique\_ptr<tm> breachDate) {
34. unique\_ptr<BreachInfo> info = make\_unique<BreachInfo>(
35. move(MakeUUID()), move(breachType), move(breachDate)
36. );
37. this->breaches->push\_back(move(info));
38. }
39. void SecurityLog::RemoveBreach(unique\_ptr<string> breachID) {
40. for (auto item = breaches->begin(); item != breaches->end(); ) {
41. if (\*(\*item)->breachID == \*breachID) {
42. item = breaches->erase(item);
43. }
44. else {
45. ++item;
46. }
47. }
48. }
49. //
50. // Класс Application
51. //
52. Application::Application(unique\_ptr<string> appName, unique\_ptr<SecurityLog> appLog) {
53. if (appName == nullptr)
54. appName = make\_unique<string>("ERROR-NAME");
55. this->appInfo = make\_unique<AppInfo>(move(MakeUUID()), move(appName));
56. if (appLog == nullptr)
57. appLog = make\_unique<SecurityLog>();
58. this->appLog = move(appLog);
59. cout << "Создан объект класса Application\n";
60. }
61. unique\_ptr<AppInfo> Application::GetAppInfo() {
62. auto info = make\_unique<AppInfo>();
63. info->appID = make\_unique<string>(\*this->appInfo->appID);
64. info->appName = make\_unique<string>(\*this->appInfo->appName);
65. return info;
66. }
67. unique\_ptr<vector<unique\_ptr<BreachInfo>>> Application::GetAppBreaches(unique\_ptr<string> breachType) {
68. return this->appLog->GetBreaches(move(breachType));
69. }
70. void Application::AddAppBreach(unique\_ptr<string> breachType, unique\_ptr<tm> breachDate) {
71. this->appLog->AddBreach(move(breachType), move(breachDate));
72. }
73. void Application::RemoveAppBreach(unique\_ptr<string> breachID) {
74. this->appLog->RemoveBreach(move(breachID));
75. }
76. //
77. // Класс User
78. //
79. User::User(unique\_ptr<string> userName) {
80. if (userName == nullptr)
81. userName = make\_unique<string>("ERROR-NAME");
82. this->userInfo = make\_unique<UserInfo>(move(MakeUUID()), move(userName));
83. this->userApps = make\_unique<vector<unique\_ptr<Application>>>();
84. cout << "Создан объект класса User\n";
85. }
86. unique\_ptr<UserInfo> User::GetUserInfo() {
87. auto info = make\_unique<UserInfo>();
88. info->userID = make\_unique<string>(\*this->userInfo->userID);
89. info->userName = make\_unique<string>(\*this->userInfo->userName);
90. cout << "Возвращена информация о пользователе\n";
91. return info;
92. }
93. unique\_ptr<vector<unique\_ptr<AppInfo>>> User::GetUserAppsInfo() {
94. auto info = make\_unique<vector<unique\_ptr<AppInfo>>>();
95. for (auto item = this->userApps->begin(); item != this->userApps->end(); ) {
96. info->push\_back(move((\*item)->GetAppInfo()));
97. ++item;
98. }
99. cout << "Возвращена информация о приложениях пользователя\n";
100. return info;
101. }
102. void User::AddApp(unique\_ptr<Application> app) {
103. this->userApps->push\_back(move(app));
104. cout << "Добавлено приложение\n";
105. }
106. void User::RemoveApp(unique\_ptr<string> appID) {
107. for (auto item = this->userApps->begin(); item != this->userApps->end(); ) {
108. if (\*(\*item)->GetAppInfo()->appID == \*appID) {
109. item = userApps->erase(item);
110. cout << "Удалено приложение\n";
111. return;
112. }
113. else {
114. ++item;
115. }
116. }
117. cout << "Не найдено приложение с заданным ID\n";
118. }
119. void User::AddAppBreach(unique\_ptr<string> appID, unique\_ptr<string> breachType, unique\_ptr<tm> breachDate) {
120. for (auto item = this->userApps->begin(); item != this->userApps->end(); ) {
121. if (\*(\*item)->GetAppInfo()->appID == \*appID) {
122. (\*item)->AddAppBreach(move(breachType), move(breachDate));
123. break;
124. }
125. else {
126. ++item;
127. }
128. }
129. cout << "Добавлено нарушение программы\n";
130. }
131. void User::RemoveAppBreach(unique\_ptr<string> appID, unique\_ptr<string> breachID) {
132. for (auto item = this->userApps->begin(); item != this->userApps->end(); ) {
133. if (\*(\*item)->GetAppInfo()->appID == \*appID) {
134. (\*item)->RemoveAppBreach(move(breachID));
135. cout << "Удалено нарушение программы\n";
136. return;
137. }
138. else {
139. ++item;
140. }
141. }
142. cout << "Не найдено приложение с заданным ID\n";
143. }
144. unique\_ptr<vector<unique\_ptr<BreachInfo>>> User::GetAppBreaches(unique\_ptr<string> appID, unique\_ptr<string> breachType) {
145. for (auto item = this->userApps->begin(); item != this->userApps->end(); ) {
146. if (\*(\*item)->GetAppInfo()->appID == \*appID) {
147. auto info = (\*item)->GetAppBreaches(move(breachType));
148. cout << "Возвращена информация о нарушениях выбранного приложения\n";
149. return info;
150. }
151. else {
152. ++item;
153. }
154. }
155. cout << "Не найденно приложение с указанным ID\n";
156. return nullptr;
157. }

## main.cpp

1. #include <iostream>
2. #include <locale>
3. #include "Class.h"
4. using namespace std;
5. int main() {
6. setlocale(LC\_ALL, "Russian");
7. auto type\_1 = make\_unique<string>("Вылет");
8. auto time\_1 = make\_unique<tm>();
9. time\_1->tm\_mday = 15;
10. auto type\_2 = make\_unique<string>("Черный экран");
11. auto time\_2 = make\_unique<tm>();
12. time\_2->tm\_mday = 9;
13. auto type\_3 = make\_unique<string>("Вылет");
14. auto time\_3 = make\_unique<tm>();
15. time\_3->tm\_mday = 26;
16. auto log\_1 = make\_unique<SecurityLog>();
17. auto name\_1 = make\_unique<string>("Angry Pigs");
18. auto app\_1 = make\_unique<Application>(move(name\_1), move(log\_1));
20. auto username = make\_unique<string>("Александр");
21. auto user = make\_unique<User>(move(username));
22. user->AddApp(move(app\_1));
23. auto user\_apps = user->GetUserAppsInfo();
24. user->AddAppBreach(make\_unique<string>(\*(\*user\_apps)[0]->appID), move(type\_1), move(time\_1));
25. user->AddAppBreach(make\_unique<string>(\*(\*user\_apps)[0]->appID), move(type\_2), move(time\_2));
26. user->AddAppBreach(make\_unique<string>(\*(\*user\_apps)[0]->appID), move(type\_3), move(time\_3));
27. auto br\_type = make\_unique<string>("Вылет");
28. auto breaches = user->GetAppBreaches(make\_unique<string>(\*(\*user\_apps)[0]->appID), move(br\_type));
30. for (auto item = breaches->begin(); item != breaches->end(); item++) {
31. cout << \*(\*item)->breachID << " -- " << \*(\*item)->breachType << " -- " << (\*item)->breachDate->tm\_mday << "\n";
32. }
33. }