rr

**МIНIСТЕРСТВО ОСВIТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**

**“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ**

**ІМЕНІ ІГОРЯ СІКОРСЬКОГО”**

**Факультет прикладної математики**

**Кафедра програмного забезпечення комп’ютерних систем**

**Лабораторна робота №6**

**з дисципліни “ Основи програмування”**

**тема “Формат даних XML”**

|  |  |  |
| --- | --- | --- |
| **Виконав**  **студент I курсу**  **групи КП-83**  **Мортіков Владислав Євгенович**  **варіант №16** |  | **Перевірив**  **“\_\_\_\_” “\_\_\_\_\_\_\_\_\_\_\_\_”  2019  р.**  **викладач**  **Гадиняк Руслан Анатолійович**  **(прізвище, ім’я, по батькові)** |

**Київ 2019**

### **Мета**

Навчитись користуватись Qt Creator і створювати консольні Qt проекти.  
Використати наслідування класів, віртуальні функції та абстрактний клас.  
Опанувати формат даних XML та навчитись його зчитувати і формувати.  
Використати поліморфізм для заміни способу роботи з файловим сховищем даних.

### **Загальні вимоги до завдання**

Кроки виконання:

1. Qt проект:
   1. Обов'язково додати у .gitignore в корені репозиторія додаткові [рядки ігнорування для Qt](https://raw.githubusercontent.com/github/gitignore/master/Qt.gitignore).
   2. [Встановити Qt i Qt Creator](http://progbase.herokuapp.com/documents/19) і відкрити Qt Creator.
   3. Створити через Qt Creator у директорії labs репозиторія новий консольний Qt проект із назвою lab6.
   4. Налаштування створеного Qt проекту будуть розміщені у файлі lab6.pro. Відкрити файл проекту і задати версію C++17.
   5. Скопіювати у створений проект ваше рішення [лабораторної роботи №5](https://progbase.herokuapp.com/modules/progbase2/tasks/lab5) (файли з кодом і директорію data, без CMakeLists.txt).
   6. (**UPD-1**). Перенести CSV файли з директорії data у data/csv. Змінити шлях до сховища даних використовуючи назву проекту, наприклад ../lab6/data/csv.  
      Це пояснюється тим, що Qt за замовчуванням збирає executable файл програми у директорії, сусідній директорії з проектом.
   7. Тепер проект має збиратись, а зібрана програма запускатись і працювати аналогічно початковому рішенню.
2. Абстрактний клас і наслідування.
   1. Перейменувати модуль storage на csv\_storage, а клас Storage на CsvStorage (лише у модулі).
   2. Створити [новий абстрактний клас](https://docs.google.com/document/d/1XI_8fQ7NscgXDIbTqmLGv-mxJRkc06Aznbte1L-IqkE/edit" \l "heading=h.j851x199itgc)Storage і публічно наслідувати CsvStorage від нього.
   3. Змінити спосіб створення сховища з використанням upcasting (**див. Додаток A**).
3. Сховище даних XML:
   1. (**UPD-1**) Створити у проекті директорію data/xml та додати у неї два XML файли (з розширенням .xml).
   2. Використати будь-який онлайн сервіс для перетворення з CSV в XML і заповнити створені файли даними основних та додаткових сутностей (приклад **див. Додаток B**).
   3. (**UPD-2**) Скопіювати модуль csv\_storage і назвати його xml\_storage з класом XmlStorage, що публічно наслідується від абстрактного класу Storage.  
      Видалити з xml\_storage все, що стосується CSV.
   4. Реалізувати XmlStorage з використанням [Qt XML класів](https://doc.qt.io/qt-5/qdomdocument.html):
      1. Додати у файл налаштувань проекту (.pro) QT += xml як перший рядок .
      2. Підключити у модулі xml\_storage заголовочний файл #include <QtXml>
   5. (**UPD-1**) При створенні об'єкта сховища замінити його тип на XmlStorage, а шлях до файлу на ../lab6/data/xml (**див. Додаток C**).

**Тексти коду програм**

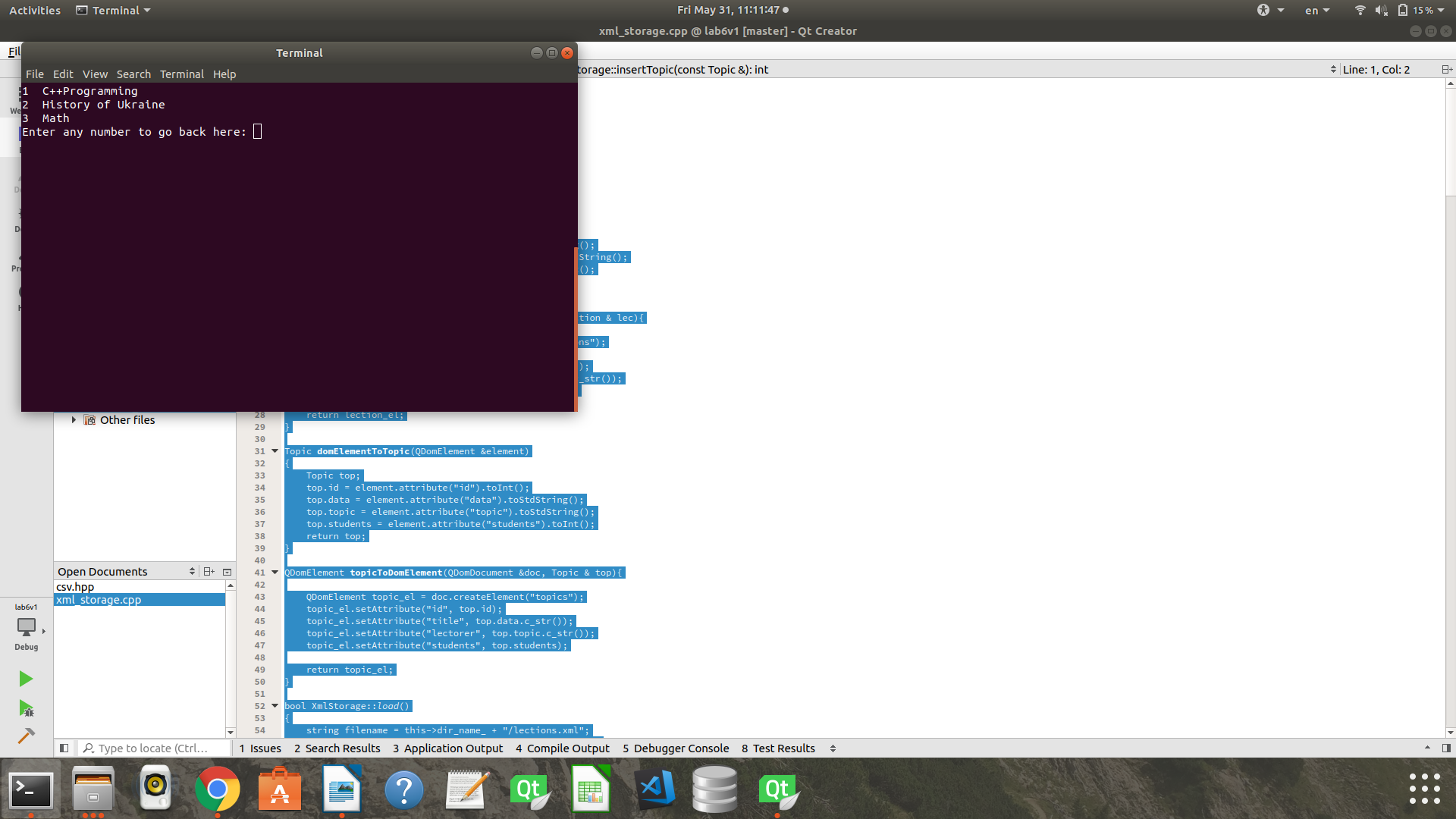
|  |
| --- |
| **xml\_storage.hpp** |
| #include <vector>  #include <string>  #include <iostream>  #include <fstream>  #include "optional.hpp"  #include "storage.hpp"  using std::string;  using std::vector;  class XmlStorage : public Storage  {  const string dir\_name\_;  vector<Lection> lections\_;  vector<Topic> topics\_;  int **getNewLectionId**();  int **getNewTopicId**();  void **print**();  public:  XmlStorage(const string &dir\_name) : dir\_name\_(dir\_name) {}  bool ***load***();  bool ***save***();  // Lections  vector<Lection> ***getAllLections***(void);  optional<Lection> ***getLectionById***(int lection\_id);  bool ***updateLection***(const Lection &Lection);  bool ***removeLection***(int lection\_id);  int ***insertLection***(const Lection &Lection);  //Topics  vector<Topic> ***getAllTopics***(void);  optional<Topic> ***getTopicById***(int topic\_id);  bool ***updateTopic***(const Topic &topic);  bool ***removeTopic***(int topic\_id);  int ***insertTopic***(const Topic &topic);  }; |

|  |
| --- |
| **xml\_storage.cpp** |
| #include "xml\_storage.hpp"  #include <QString>  #include <QDebug>  #include <QFile>  #include <QtXml>  using namespace std;  Lection **domElementToLection**(QDomElement &element)  {  Lection lec;  lec.id = element.attribute("id").toInt();  lec.title = element.attribute("title").toStdString();  lec.lectorer = element.attribute("lectorer").toStdString();  lec.students = element.attribute("students").toInt();  return lec;  }  QDomElement **lectionToDomElement**(QDomDocument &doc, Lection & lec){  QDomElement lection\_el = doc.createElement("lections");  lection\_el.setAttribute("id", lec.id);  lection\_el.setAttribute("title", lec.title.c\_str());  lection\_el.setAttribute("lectorer", lec.lectorer.c\_str());  lection\_el.setAttribute("students", lec.students);  return lection\_el;  }  Topic **domElementToTopic**(QDomElement &element)  {  Topic top;  top.id = element.attribute("id").toInt();  top.data = element.attribute("data").toStdString();  top.topic = element.attribute("topic").toStdString();  top.students = element.attribute("students").toInt();  return top;  }  QDomElement **topicToDomElement**(QDomDocument &doc, Topic & top){  QDomElement topic\_el = doc.createElement("topics");  topic\_el.setAttribute("id", top.id);  topic\_el.setAttribute("title", top.data.c\_str());  topic\_el.setAttribute("lectorer", top.topic.c\_str());  topic\_el.setAttribute("students", top.students);  return topic\_el;  }  bool XmlStorage::***load***()  {  string filename = this->dir\_name\_ + "/lections.xml";  QString q\_filename = QString::fromStdString(filename);  QFile file(q\_filename);  bool isOpened = file.*open*(QFile::ReadOnly);  if (!isOpened){  qDebug() << "CAN'T OPEN THE FILE " << q\_filename;  return false;  }  QTextStream ts(&file);  QString text = ts.readAll();  file.*close*();  QDomDocument doc;  QString errorMessage;  int errorLine;  int errorColumn;  if (!doc.setContent(text,&errorMessage,&errorLine,&errorColumn)){  qDebug() << "Error parsing XML text: " << errorMessage;  qDebug() << "at line" << errorLine << ", column " << errorColumn;  return false;  }  qDebug() << "OK!:)";  QDomElement root = doc.documentElement();  for(int i = 0; i < root.childNodes().size(); i++)  {  QDomNode node = root.childNodes().at(i);  QDomElement element = node.toElement();  Lection lec = domElementToLection(element);  this->lections\_.push\_back(lec);  }  //----------------------------------------  string topicFilename = this->dir\_name\_ + "/topics.xml";  QString q\_topicFilename = QString::fromStdString(topicFilename);  QFile fileTF(q\_topicFilename);  bool isOpenedTF = fileTF.*open*(QFile::ReadOnly);  if (!isOpenedTF){  qDebug() << "CAN'T OPEN THE FILE " << q\_topicFilename;  return false;  }  QTextStream tstf(&fileTF);  QString textTF = tstf.readAll();  fileTF.*close*();  QDomDocument docTF;  QString errorMessageTF;  int errorLineTF;  int errorColumnTF;  if (!docTF.setContent(textTF,&errorMessageTF,&errorLineTF,&errorColumnTF)){  qDebug() << "Error parsing XML text: " << errorMessageTF;  qDebug() << "at line" << errorLineTF << ", column " << errorColumnTF;  return false;  }  qDebug() << "OK!:)";  QDomElement rootTF = docTF.documentElement();  for(int i = 0; i < rootTF.childNodes().size(); i++)  {  QDomNode nodeTF = rootTF.childNodes().at(i);  QDomElement elementTF = nodeTF.toElement();  Topic top = domElementToTopic(elementTF);  this->topics\_.push\_back(top);  }  return true;  }  bool XmlStorage::***save***()  {  QDomDocument doc;  QDomElement root = doc.createElement("lections");  for(Lection & lec : this->lections\_){  QDomElement lection\_el = lectionToDomElement(doc, lec);  root.appendChild(lection\_el);  }  doc.appendChild(root);  QString xml\_text = doc.toString(4);  string filename = this->dir\_name\_ + "/lections.xml";  QString q\_filename = QString::fromStdString(filename);  QFile file(q\_filename);  bool isOpened = file.*open*(QFile::WriteOnly);  if (!isOpened){  qDebug() << "CAN'T OPEN THE FILE " << q\_filename;  return false;  }  QTextStream ts(&file);  ts << xml\_text;  file.*close*();  //--------------------------------------------  QDomDocument docT;  QDomElement rootT = docT.createElement("topics");  for(Topic & top : this->topics\_){  QDomElement topic\_el = topicToDomElement(doc, top);  rootT.appendChild(topic\_el);  }  docT.appendChild(rootT);  QString xml\_textT = doc.toString(4);  string filenameT = this->dir\_name\_ + "/topics.xml";  QString q\_filenameT = QString::fromStdString(filenameT);  QFile fileT(q\_filenameT);  bool isOpenedT = fileT.*open*(QFile::WriteOnly);  if (!isOpenedT){  qDebug() << "CAN'T OPEN THE FILE " << q\_filenameT;  return false;  }  QTextStream tsT(&file);  tsT << xml\_textT;  file.*close*();  return true;  }  int XmlStorage::**getNewLectionId**()  {  int max = 0;  for (Lection lec : this->lections\_)  {  if (max < lec.id)  max = lec.id;  }  return max + 1;  }  int XmlStorage::**getNewTopicId**()  {  int max = 0;  for (Topic top : this->topics\_)  {  if (max < top.id)  max = top.id;  }  return max + 1;  }  vector<Lection> XmlStorage::***getAllLections***()  {  return this->lections\_;  }  optional<Lection> XmlStorage::***getLectionById***(int lection\_id)  {  for (Lection &lec : this->lections\_)  {  if (lec.id == lection\_id)  {  return lec;  }  }  return nullopt;  }  bool XmlStorage::***updateLection***(const Lection &lection)  {  for (int i = 0; i < lections\_.size(); i++)  {  if (lections\_[i].id == lection.id)  {  lections\_[i] = lection;  return 1;  }  }  return 0;  }  bool XmlStorage::***removeLection***(int lection\_id)  {  for (int i = 0; i < lections\_.size(); i++)  {  if (lections\_[i].id == lection\_id)  {  lections\_.erase(lections\_.begin() + i);  return 1;  }  }  return 0;  }  int XmlStorage::***insertLection***(const Lection &lection)  {  Lection lec = lection;  lec.id = getNewLectionId();  lections\_.push\_back(lec);  return lec.id;  }  vector<Topic> XmlStorage::***getAllTopics***()  {  return this->topics\_;  }  optional<Topic> XmlStorage::***getTopicById***(int lection\_id)  {  for (Topic &top : this->topics\_)  {  if (top.id == lection\_id)  {  return top;  }  }  return nullopt;  }  bool XmlStorage::***updateTopic***(const Topic &topic)  {  for (int i = 0; i < topics\_.size(); i++)  {  if (topics\_[i].id == topic.id)  {  topics\_[i] = topic;  return 1;  }  }  return 0;  }  bool XmlStorage::***removeTopic***(int topic\_id)  {  for (int i = 0; i < topics\_.size(); i++)  {  if (topics\_[i].id == topic\_id)  {  topics\_.erase(topics\_.begin() + i);  return 1;  }  }  return 0;  }  int XmlStorage::***insertTopic***(const Topic &topic)  {  Topic top = topic;  top.id = getNewTopicId();  topics\_.push\_back(top);  return top.id;  } |

|  |
| --- |
| **csv\_storage.hpp** |
| #include <vector>  #include <string>  #include <iostream>  #include <fstream>  #include "optional.hpp"  #include "storage.hpp"  #include "csv.hpp"  using std::string;  using std::vector;  class CsvStorage : public Storage  {  const string dir\_name\_;  vector<Lection> lections\_;  vector<Topic> topics\_;  int **getNewLectionId**();  int **getNewTopicId**();  void **print**();  public:  CsvStorage(const string &dir\_name) : dir\_name\_(dir\_name) {}  bool ***load***();  bool ***save***();  // Lections  vector<Lection> ***getAllLections***(void);  optional<Lection> ***getLectionById***(int lection\_id);  bool ***updateLection***(const Lection &Lection);  bool ***removeLection***(int lection\_id);  int ***insertLection***(const Lection &Lection);  //Topics  vector<Topic> ***getAllTopics***(void);  optional<Topic> ***getTopicById***(int topic\_id);  bool ***updateTopic***(const Topic &topic);  bool ***removeTopic***(int topic\_id);  int ***insertTopic***(const Topic &topic); |

|  |
| --- |
| **csv\_storage.cpp** |
| #include "csv\_storage.hpp"  using namespace std;  void **writeAllToFile**(string const &filename, string const &text);  bool CsvStorage::***load***()  {  string lections\_filename = this->dir\_name\_ + "/lections.csv";  ifstream file;  file.open(lections\_filename, ios::in);  if (!file.good())  {  cerr << "Can't open the file: " << lections\_filename << endl;  abort();  }  string tmp, str;  while (getline(file, tmp))  {  str += tmp + '\n';  tmp.clear();  }  CsvTable lections\_table = Csv::createTableFromString(str);  for (CsvRow &row : lections\_table)  {  Lection lec;  if (row.size() > 0)  lec.id = stoi(row[0]);  if (row.size() > 1)  lec.title = row[1];  if (row.size() > 2)  lec.lectorer = row[2];  if (row.size() > 3)  lec.students = stoi(row[3]);  this->lections\_.push\_back(lec);  }  file.close();  // ----------------------------------  string topics\_filename = this->dir\_name\_ + "/topics.csv";  file.open(topics\_filename, ios::in);  if (!file.good())  {  cerr << "Can't open the file: " << topics\_filename << endl;  abort();  }  str.clear();  while (getline(file, tmp))  {  str += tmp + '\n';  tmp.clear();  }  CsvTable topic\_table = Csv::createTableFromString(str);  for (CsvRow &row : topic\_table)  {  Topic top;  if (row.size() > 0)  top.id = stoi(row[0]);  if (row.size() > 1)  top.data = row[1];  if (row.size() > 2)  top.topic = row[2];  if (row.size() > 3)  top.students = stoi(row[3]);  this->topics\_.push\_back(top);  }  file.close();  return true;  }  bool CsvStorage::***save***()  {  CsvTable table;  for (Lection lec : lections\_)  {  CsvRow row;  row.push\_back(to\_string(lec.id));  row.push\_back(lec.title);  row.push\_back(lec.lectorer);  row.push\_back(to\_string(lec.students));  table.push\_back(row);  };  string filename = "lections.csv";  ofstream output;  output.open(dir\_name\_ + filename);  if (!output.is\_open())  {  cout << "no such file: " << dir\_name\_;  output.close();  return 0;  }  output << Csv::createStringFromTable(table);  output.close();  ////////////////  CsvTable table1;  for (Topic top : topics\_)  {  CsvRow row;  row.push\_back(to\_string(top.id));  row.push\_back(top.data);  row.push\_back(top.topic);  row.push\_back(to\_string(top.students));  table1.push\_back(row);  }  filename = "topics.csv";  output.open(dir\_name\_ + filename);  if (!output.is\_open())  {  cout << "no such file: " << dir\_name\_;  output.close();  return 0;  }  output << Csv::createStringFromTable(table1);  output.close();  return 1;  }  int CsvStorage::**getNewLectionId**()  {  int max = 0;  for (Lection lec : this->lections\_)  {  if (max < lec.id)  max = lec.id;  }  return max + 1;  }  int CsvStorage::**getNewTopicId**()  {  int max = 0;  for (Topic top : this->topics\_)  {  if (max < top.id)  max = top.id;  }  return max + 1;  }  vector<Lection> CsvStorage::***getAllLections***()  {  return this->lections\_;  }  optional<Lection> CsvStorage::***getLectionById***(int lection\_id)  {  for (Lection &lec : this->lections\_)  {  if (lec.id == lection\_id)  {  return lec;  }  }  return nullopt;  }  bool CsvStorage::***updateLection***(const Lection &lection)  {  for (int i = 0; i < lections\_.size(); i++)  {  if (lections\_[i].id == lection.id)  {  lections\_[i] = lection;  return 1;  }  }  return 0;  }  bool CsvStorage::***removeLection***(int lection\_id)  {  for (int i = 0; i < lections\_.size(); i++)  {  if (lections\_[i].id == lection\_id)  {  lections\_.erase(lections\_.begin() + i);  return 1;  }  }  return 0;  }  int CsvStorage::***insertLection***(const Lection &lection)  {  Lection lec = lection;  lec.id = getNewLectionId();  lections\_.push\_back(lec);  return lec.id;  }  vector<Topic> CsvStorage::***getAllTopics***()  {  return this->topics\_;  }  optional<Topic> CsvStorage::***getTopicById***(int lection\_id)  {  for (Topic &top : this->topics\_)  {  if (top.id == lection\_id)  {  return top;  }  }  return nullopt;  }  bool CsvStorage::***updateTopic***(const Topic &topic)  {  for (int i = 0; i < topics\_.size(); i++)  {  if (topics\_[i].id == topic.id)  {  topics\_[i] = topic;  return 1;  }  }  return 0;  }  bool CsvStorage::***removeTopic***(int topic\_id)  {  for (int i = 0; i < topics\_.size(); i++)  {  if (topics\_[i].id == topic\_id)  {  topics\_.erase(topics\_.begin() + i);  return 1;  }  }  return 0;  }  int CsvStorage::***insertTopic***(const Topic &topic)  {  Topic top = topic;  top.id = getNewTopicId();  topics\_.push\_back(top);  return top.id;  }  void **writeAllToFile**(string const &filename, string const &text)  {  ofstream file;  file.open(filename);  if (!file.good())  {  cerr << "The file cannot open for write: " << filename << endl;  abort();  }  file << text;  file.close();  } |

**￼Приклади результатів**

****

**Висновки**

Виконавши дану лабораторну роботу ми навчилися користуватись Qt Creator і створювати консольні Qt проекти.  
Використати наслідування класів, віртуальні функції та абстрактний клас.  
Опанували формат даних XML та навчитись його зчитувати і формувати.  
Використали поліморфізм для заміни способу роботи з файловим сховищем даних.