III. Программное обеспечение для объектно-ориентированного программирования и разработки приложений (Kdevelop, Lazarus, Gambas). Практические задания

Данный набор практик ставит целью познакомить слушателей с «живым» созданием ПО в каждой из трех изучаемых сред разработки.

1. Практика работы с Kdevelop и языком программирования C++

1. Убедитесь, что вы вошли в систему с именем пользователя (login) **admin**. После этого из главного меню **КДЕ-Прочие-Разработка-KDevelop-Среда разработки на C/C++** (Kdevelop: C/C++) запустите среду разработки KDevelop (Рис. 1).

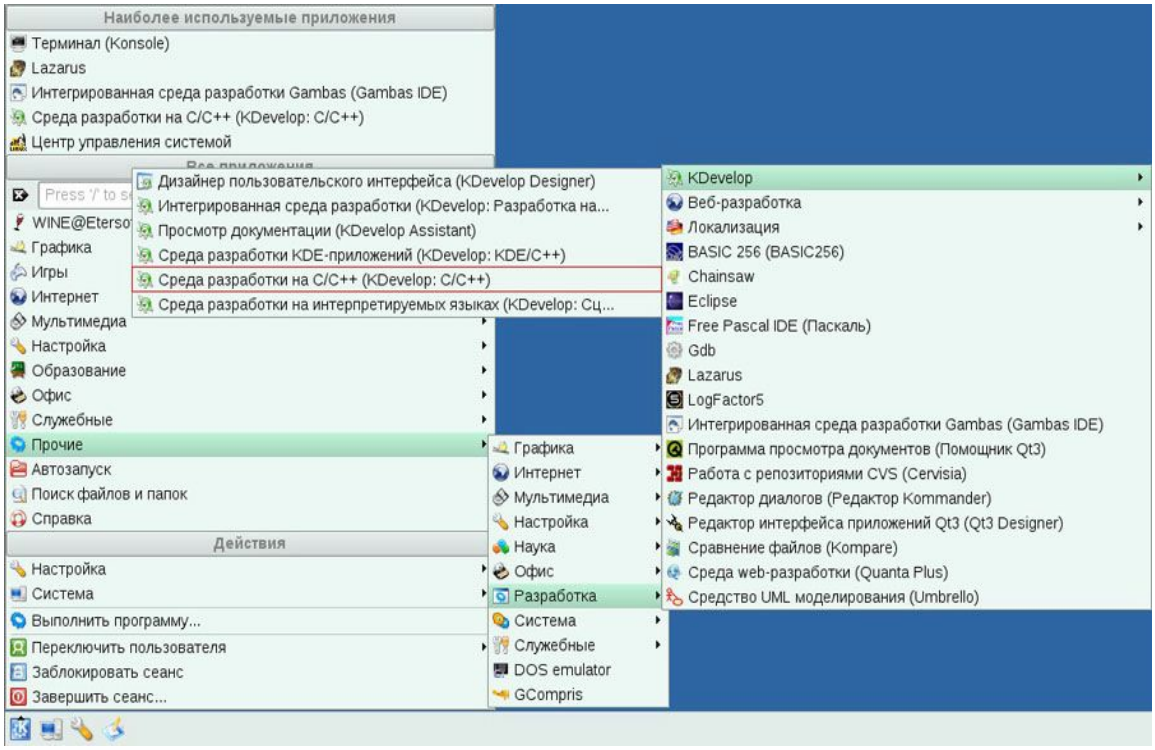

Рис. 1

2. Откроется рабочее пространство (workspace) пакета с диалоговым окном **«Совет дня».** В последнем снимите галочку **Показывать советы при запуске** и нажмите **Закрыть** (Рис. 2).

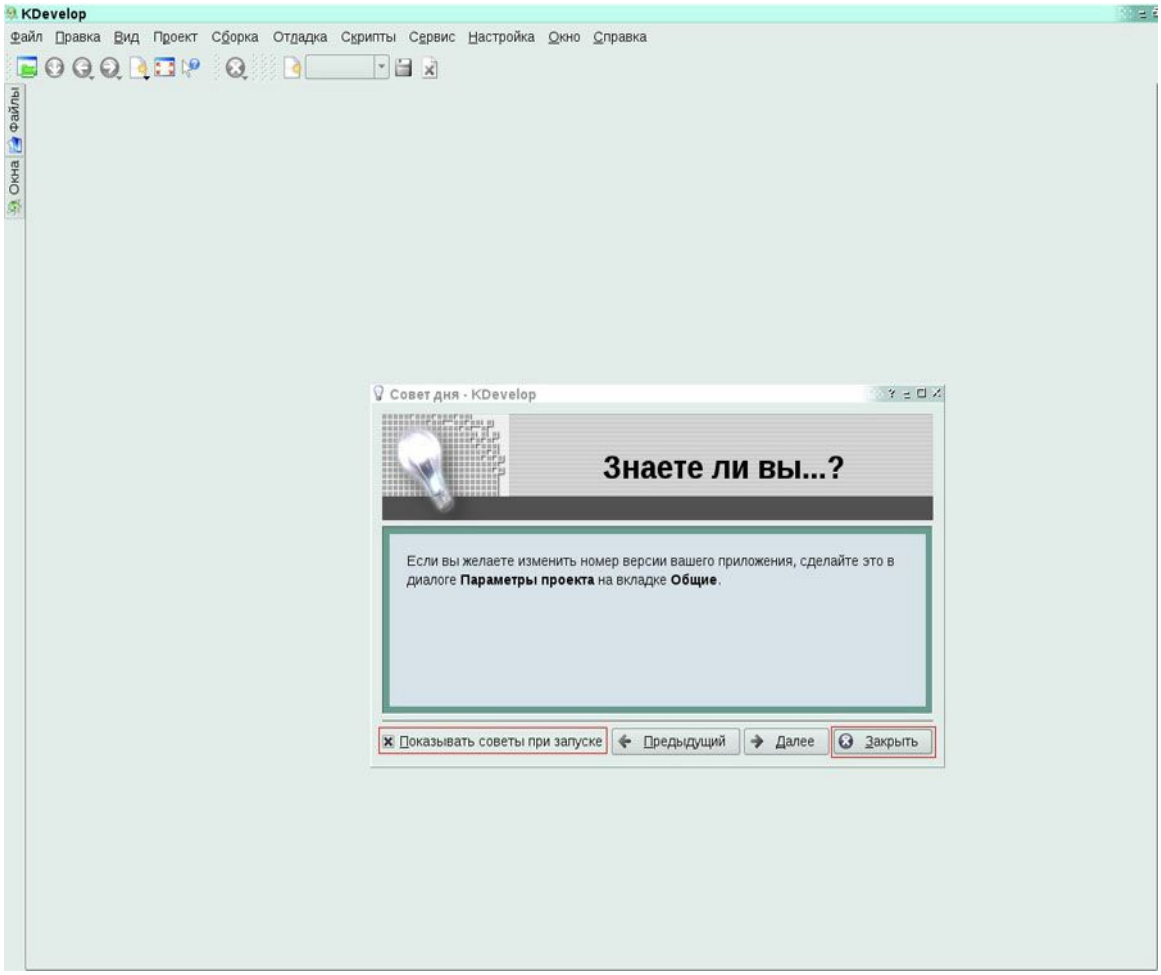

Рис. 2

3. Поскольку мы планируем начать новый консольный проект выберите в главном меню пункты **Проект-Создать проект** (Рис. 3).

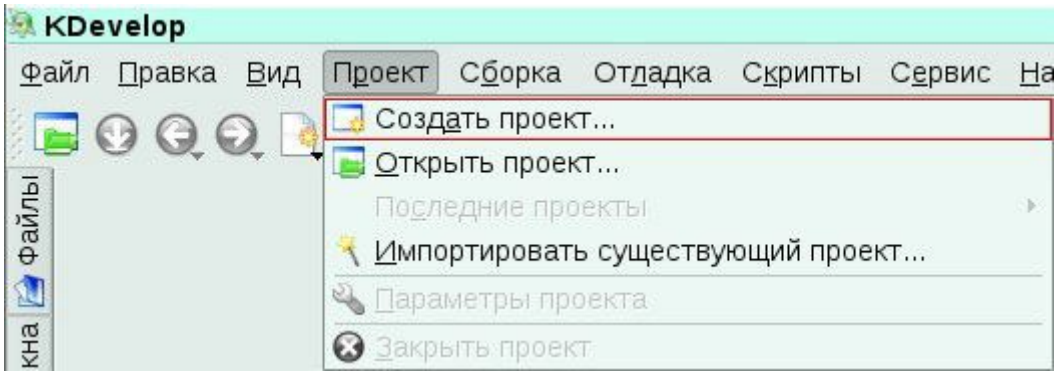

Рис. 3

4. Вы увидите окно **«Новый проект»**, в нем:

- выберите тип проекта - распахните узел *C++* и отметьте под-узел **Простая программа Hello world**.

- назовите свое приложение **ClassDir_KDevelop**, для чего введите это имя в поле **Имя приложения**
- оставьте предлагаемое **Расположение проекта** без изменений (**/home/admin/**)
- убедитесь, что домашней папкой нового проекта станет **/home/admin/ClassDir_KDevelop**
- нажмите **Вперед** (Рис. 4)

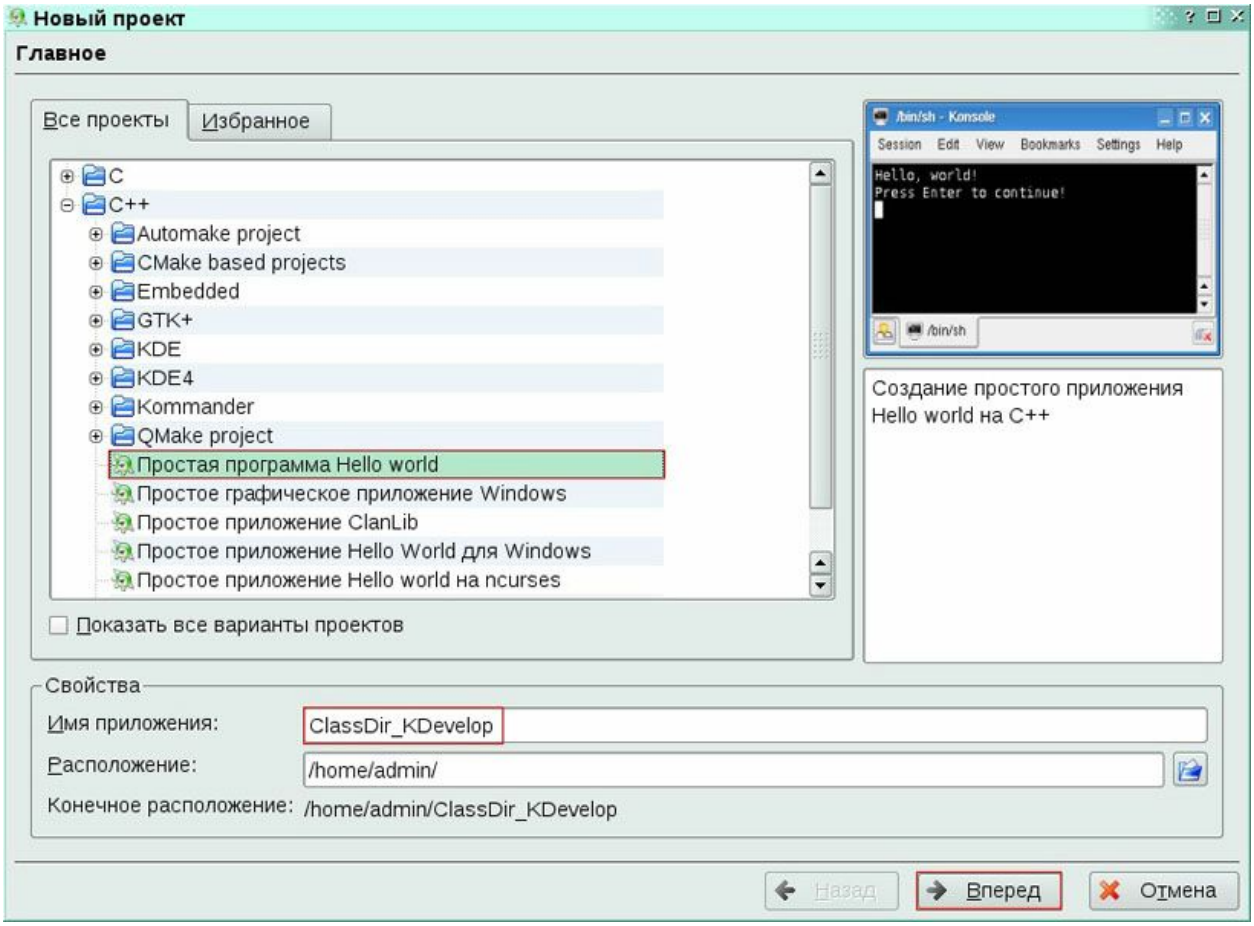

Рис. 4

5. На закладке **«Параметры проекта»** введите свое имя (например, **Peter**) в поле **Автор**. Все остальные настройки оставьте «как есть». Нажмите **Вперед** (Рис. 5).

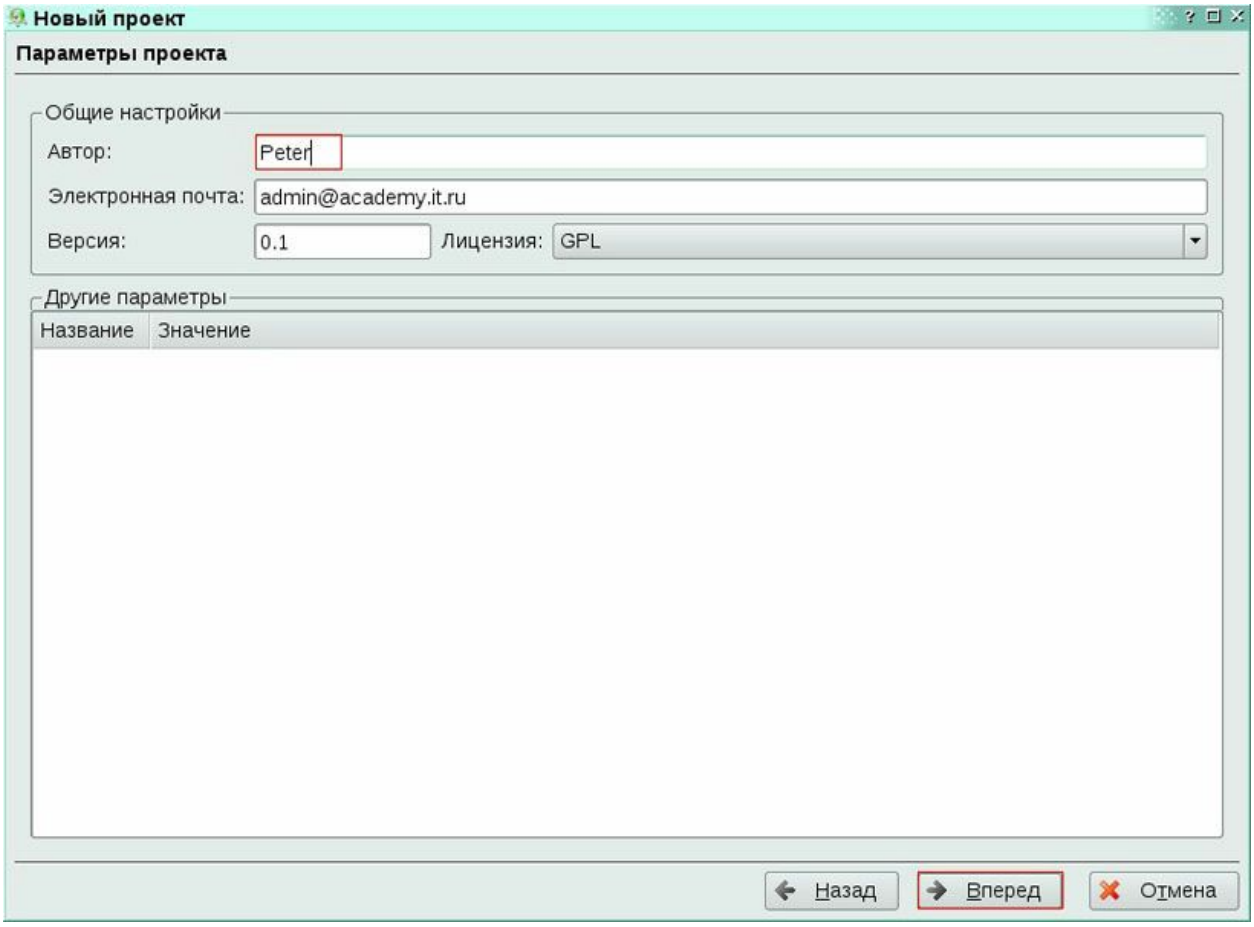

Рис. 5

6. Для нашей простой программы система управления версиями нам не понадобится, а поэтому на одноименной закладке убедитесь что в выпадающем списке выбрано **Не используется** (это значение по умолчанию) и нажмите **Вперед** (Рис. 6)

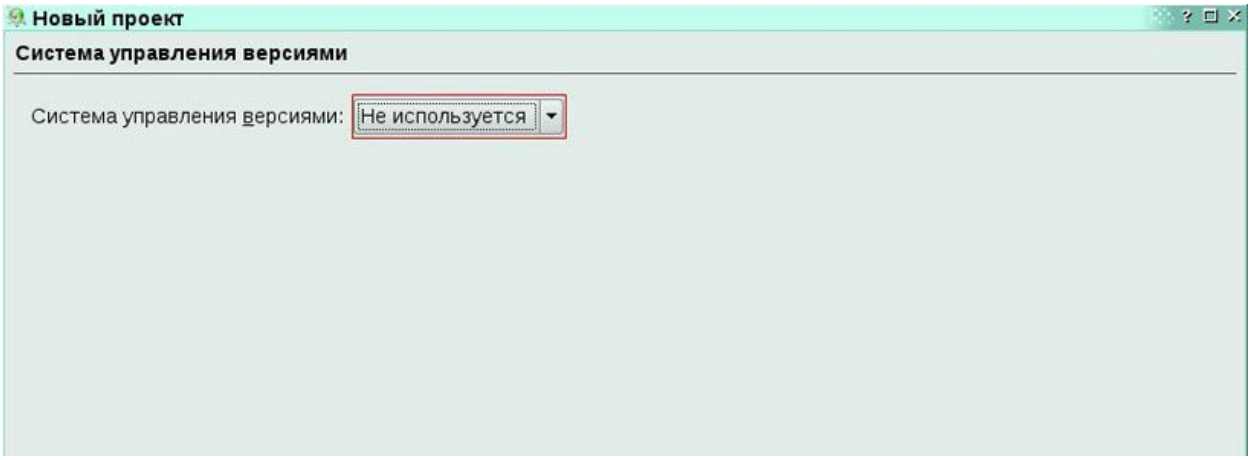

Рис. 6

7. В следующей закладке **«Шаблон для файлов .h»** нажатием кнопки **Вперед** примите шаблон предлагаемый по умолчанию (Рис. 7).

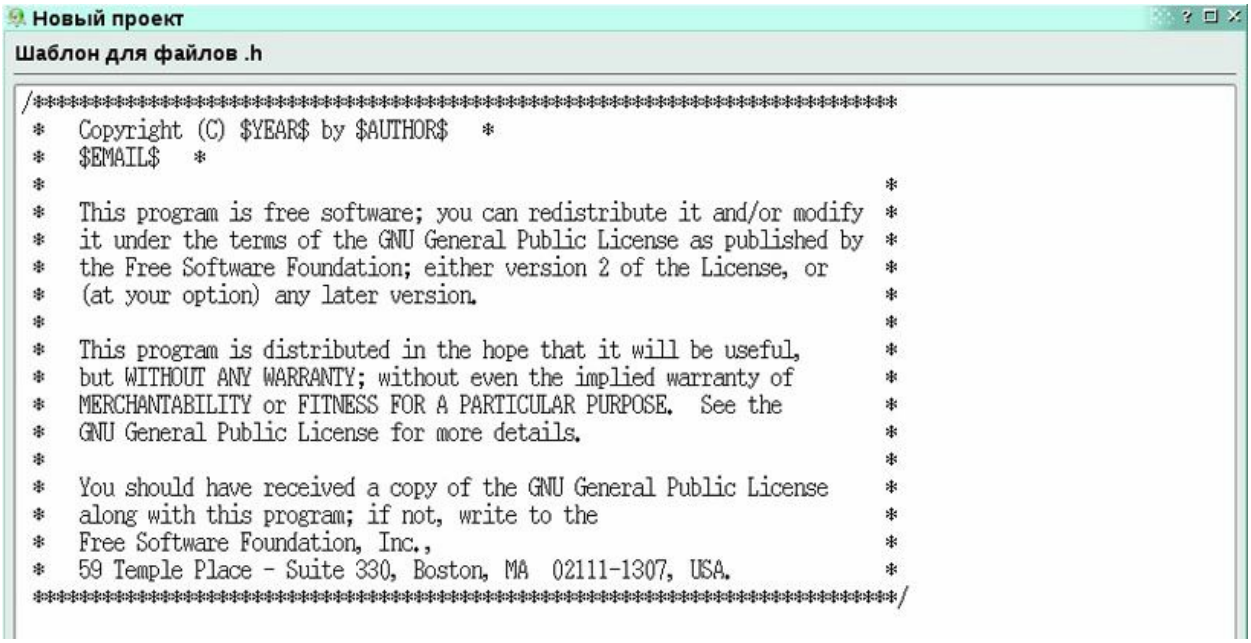

Рис. 7

8. На закладке последней, **«Шаблон для файлов .cpp»** поступите так же и просто нажмите кнопку **Готово** (Рис. 8).

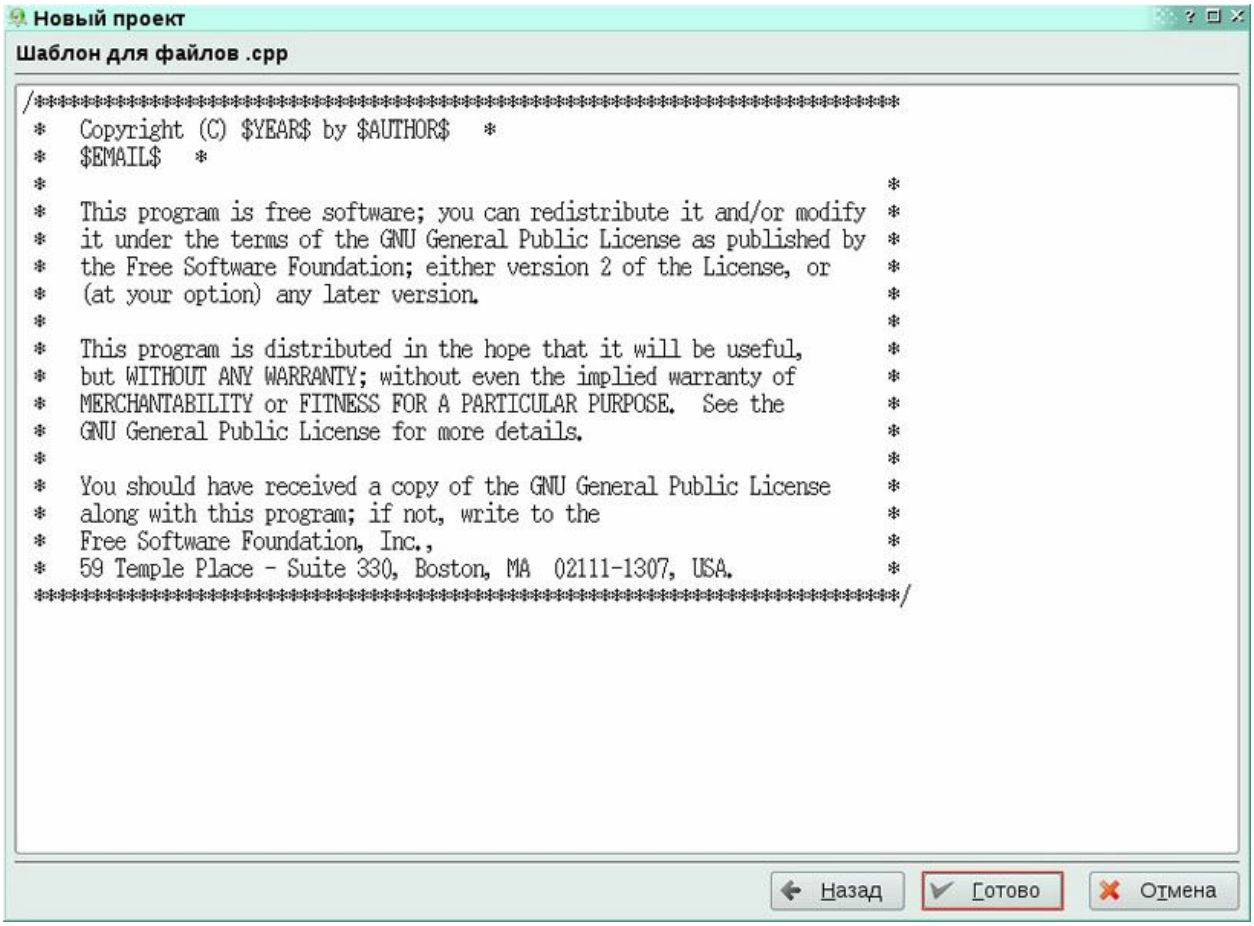

Рис. 8

9. Откроется редактор с кодом единственного (пока) файла нашего проекта - **classdir_kdevelop.cpp.** Включите нумерацию строк в редакторе. Это позволит легче соотносить текст данной практической работы с текстом создаваемого проекта. Для этого выберите в главном меню **Вид-Показать номера строк** (Рис. 9).

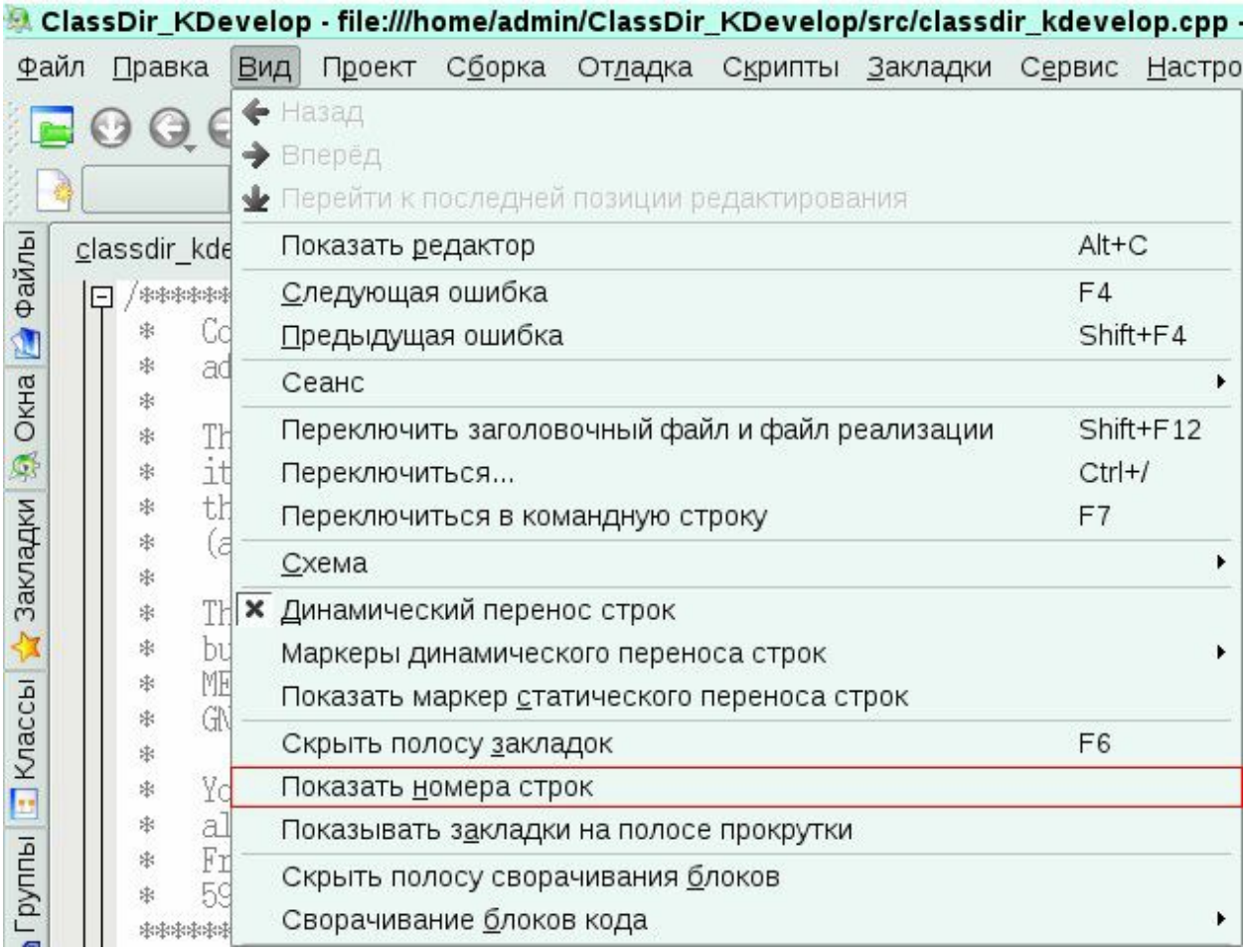

Рис. 9

10. Согласно дизайну нашего проекта нам нужен новый файл, в котором мы реализуем класс **Dir**. Для создания нового класса (и файла «под» него) выберите в главном меню **Проект-Создать класс…(**Рис. 10).

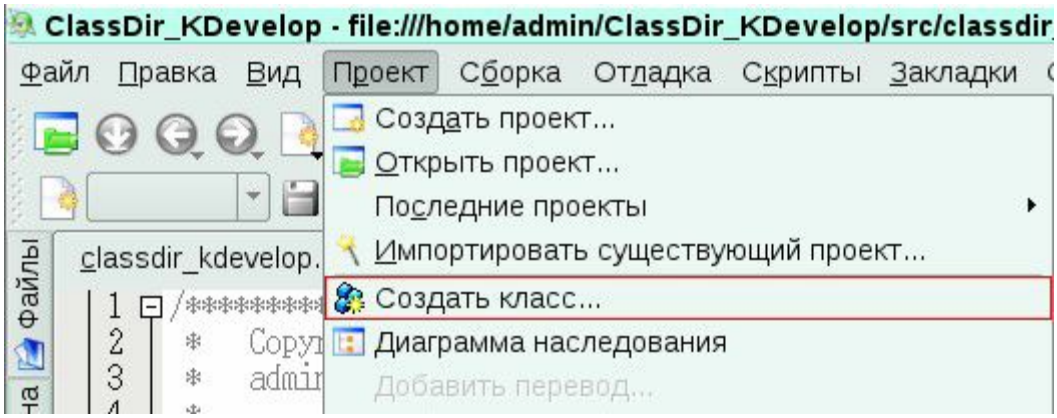

Рис. 10

11. Появляется окно «Новый класс» содержащее много параметров, по которым будет генерироваться исходный текст нашего класса. Нам достаточно указать только его имя и оставить все остальные настройки без изменений. Введите имя нашего класса - **Dir** - в поле **Имя** и нажмите кнопку **OK** (Рис. 11).

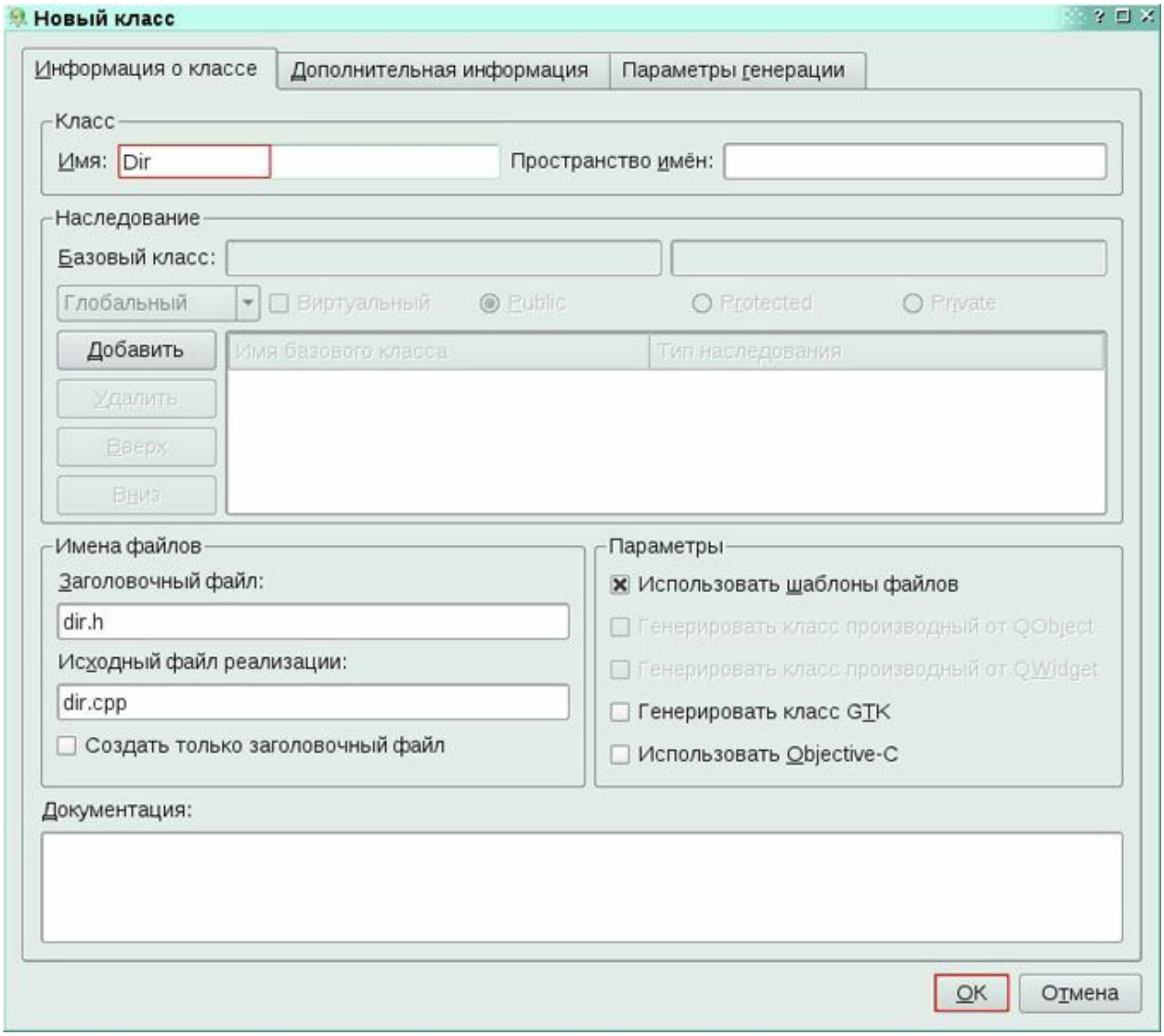

Рис. 11

12. В C++ принято разделять *описание* класса и его *реализацию* по двум файлам. Имя первого файла строится по шаблону ***<имя_класса>*.h**, второго - ***<имя_класса>*.cpp.** Первые называют заголовочными файлами, вторые - файлами реализации. В первых описывается только состав класса (поля, их типы, методы и т.д.) но не его реализация, которая как раз и выносится в файлы второго типа. Если сказать упрощенно в **.h**-файле должен содержаться весь текст нашего класса, **кроме** тел его методов (однако *описания* методов включаются именно сюда); в **.cpp**-файле, соответственно, будут только тела методов. Согласно такому подходу шаг предыдущий добавил нам в проект два файла - dir.h и dir.cpp. В настоящий момент оба они, а так же файл главного метода Main открыты в своих закладках редактора. Заголовки этих

закладок эти идут поверх окна редактора, и переключаться между ними можно щелчком мыши непосредственно по заголовку. Давайте начнем с описания нашего класса а для этого щелкните по закладке dir.h и способом, указанным в шаге 9 нашей практической работы включите нумерацию строк и для этой закладки.

Полученный результат представлен на Рис. 12. На нем же приведен исходный текст этого файла как он был сгенерирован средой.

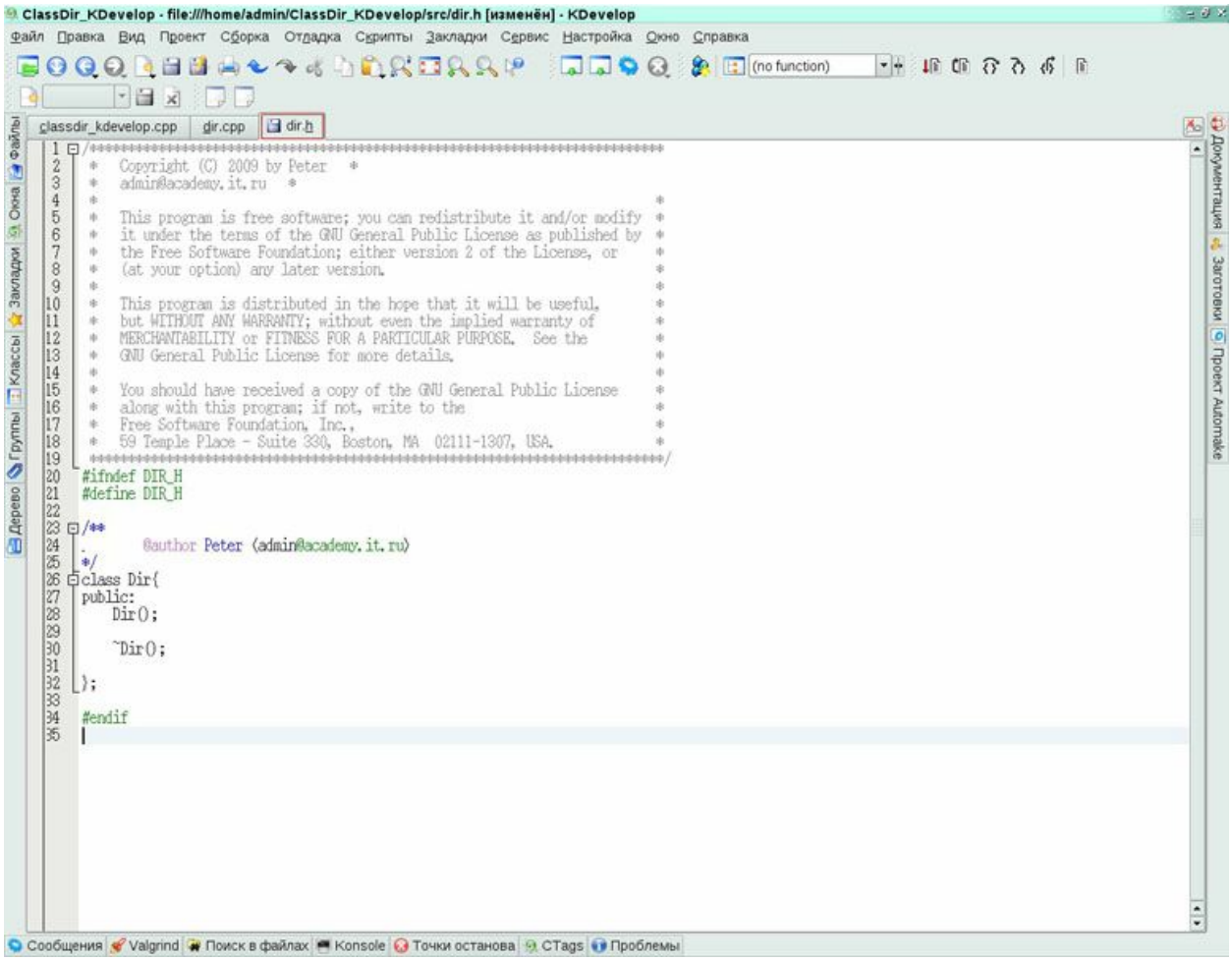

Рис. 12

С целью возможности «отката» к этой отправной точке начальный текст файла **dir.h** приводится здесь:

```
      /**************************************************************************
****
       *   Copyright (C) 2009 by Peter   *
       *   admin@academy.it.ru   *
       *
*
       *   This program is free software; you can redistribute it and/or
modify  *
       *   it under the terms of the GNU General Public License as published
by  *
       *   the Free Software Foundation; either version 2 of the License, or
*
       *   (at your option) any later version.
*
```

```
 *
*
 *   This program is distributed in the hope that it will be useful,
*
 *   but WITHOUT ANY WARRANTY; without even the implied warranty of
*
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
*
 *   GNU General Public License for more details.
*
 *
*
 *   You should have received a copy of the GNU General Public License
*
 *   along with this program; if not, write to the
*
 *   Free Software Foundation, Inc.,
*
 *   59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.
*
**************************************************************************/
#ifndef DIR_H
#define DIR_H

/**
      @author Peter <admin@academy.it.ru>
*/
class Dir{
public:
    Dir();

    ~Dir();

};

#endif
```

13. В настоящий момент описание класса говорит о том, что **Dir** не имеет никаких полей, но имеет 2 метода - конструктор и деструктор. Первый, как мы знаем, вызывается в момент создания экземпляра класса. Второй ему полностью противоположен и вызывается в момент разрушения экземпляра существующего в момент когда необходимость в нем отпадает. Реализация деструктора нужна только если мы хотим какого-то особого поведения нашего класса и его экземпляров в этой заключительной стадии их работы. В нашем случае никакого особого поведения не требуется, мы можем просто оставить деструктор пустым.

Согласно намеченной в разделе «Постановка задачи» дизайн-схеме нашего проекта внесите описание трех полей нашего класса. Такая декларация должна начинаться после открывающей фигурной скобки, но **до** ключевого слова **public**, т.к. соблюдая принципы ООП, мы собираемся предоставить доступ к полям только самому классу **Dir** и никому более. Т.о. сдвинем текущую линию 27 на одну вниз и введем описания полей:

```
class Dir{
string _name;
```

```
    int _numOfChild;
    bool _isOwner;
    public:
        Dir();

        ~Dir();

    };
```

14. В файле **.h** нам осталось лишь объявить (но не реализовать) оба метода нашего класса - конструктор и GetInfo. Конструктор уже объявлен, однако его объявление нам не подходит. Согласно нашему плану нам требуется конструктор с тремя аргументами. Текущий же конструктор аргументов не принимает. Поэтому заменим 31-ю строку такой:

```
Dir(string name,int numOfChild,bool isOwner);
```

И, наконец, в следующей строке объявим второй метод:

```
void GetInfo();
```

15. Мы знаем, что последний метод будет работать с консолью, а объявления классов/методов для работы с ней находятся в заголовочном файле iostream.h. Кроме того, один из аргументов нашего конструктора имеет тип string. А это - не элементарный тип, т.е. он не «встроен» в сам язык C++. В отличии от истинно элементарных типов (int, char, bool и пр.) это - класс и мы должны включить его определение в нашу программу. Все вышесказанное выражается участком кода

```
#include <iostream>
#include <string>

using namespace std;
```

который надо вставить ДО объявления класса **Dir.** Строка 22 выглядит подходящей для этой операции.

16. Итоговый текст файла **dir.h** будет таким:

```
/*************************************************************************
****
 *   Copyright (C) 2009 by Peter   *
 *   admin@academy.it.ru   *
 *
*
 *   This program is free software; you can redistribute it and/or
modify  *
 *   it under the terms of the GNU General Public License as published
by  *
 *   the Free Software Foundation; either version 2 of the License, or
*
 *   (at your option) any later version.
*
```

```
 *
*
 *   This program is distributed in the hope that it will be useful,
*
 *   but WITHOUT ANY WARRANTY; without even the implied warranty of
*
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
*
 *   GNU General Public License for more details.
*
 *
*
 *   You should have received a copy of the GNU General Public License
*
 *   along with this program; if not, write to the
*
 *   Free Software Foundation, Inc.,
*
 *   59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.
*
**************************************************************************/
#ifndef DIR_H
#define DIR_H
#include <iostream>
#include <string>

using namespace std;

/**
      @author Peter <admin@academy.it.ru>
*/
class Dir{
string _name;
int _numOfChild;
bool _isOwner;
public:
    Dir(string name,int numOfChild,bool isOwner);
    void GetInfo();
    ~Dir();

};

#endif
```

А непосредственно в редакторе он будет выглядеть так, как показано на Рис. 13.

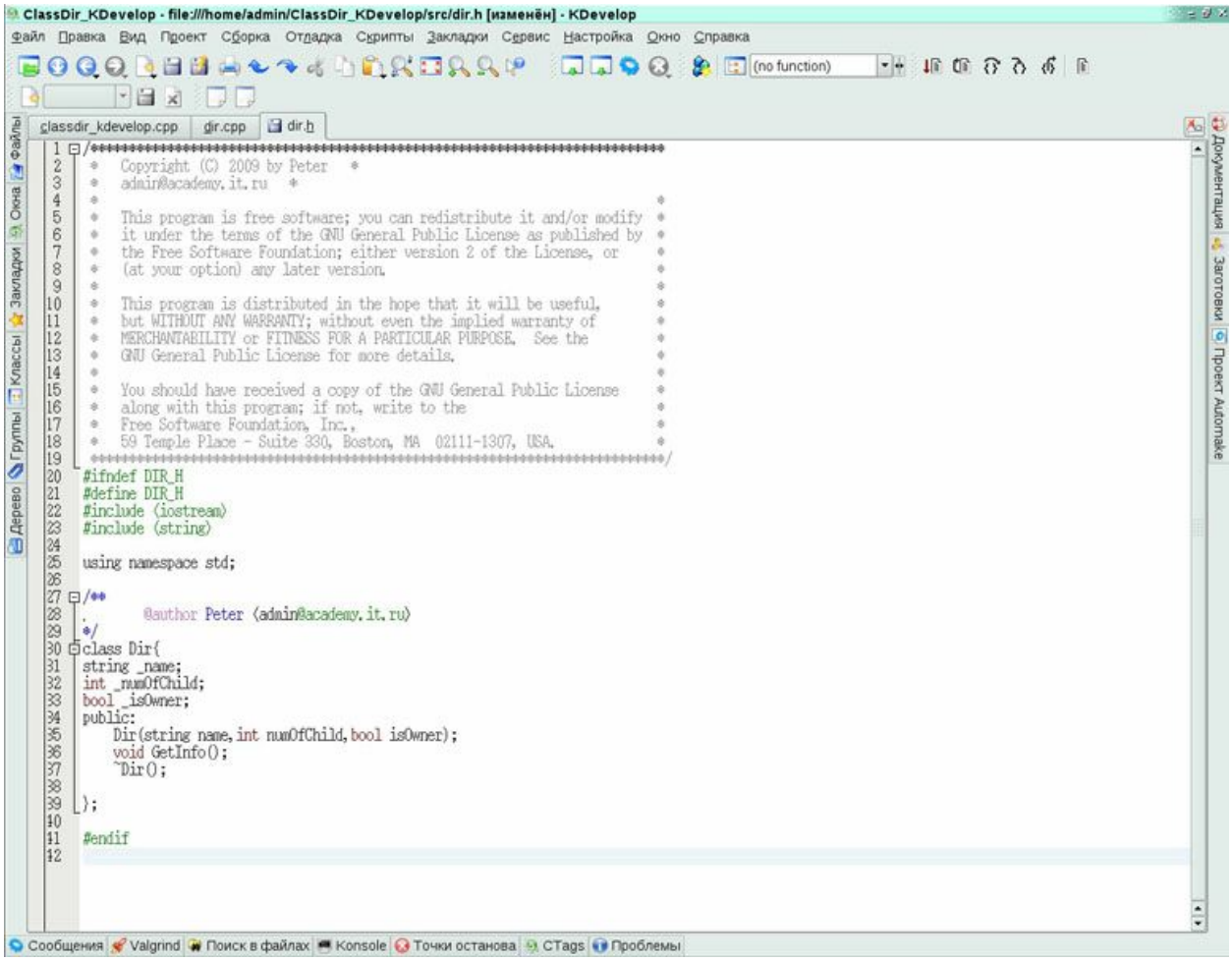

Рис. 13

17. Перейдем к файлу реализации, dir.cpp для чего щелкнем по заголовку его закладки и способом, указанным в шаге 9 нашей практической работы включим нумерацию строк и для этой закладки. Полученный результат представлен на Рис. 14. На нем же приведен исходный текст этого файла как он был сгенерирован средой.

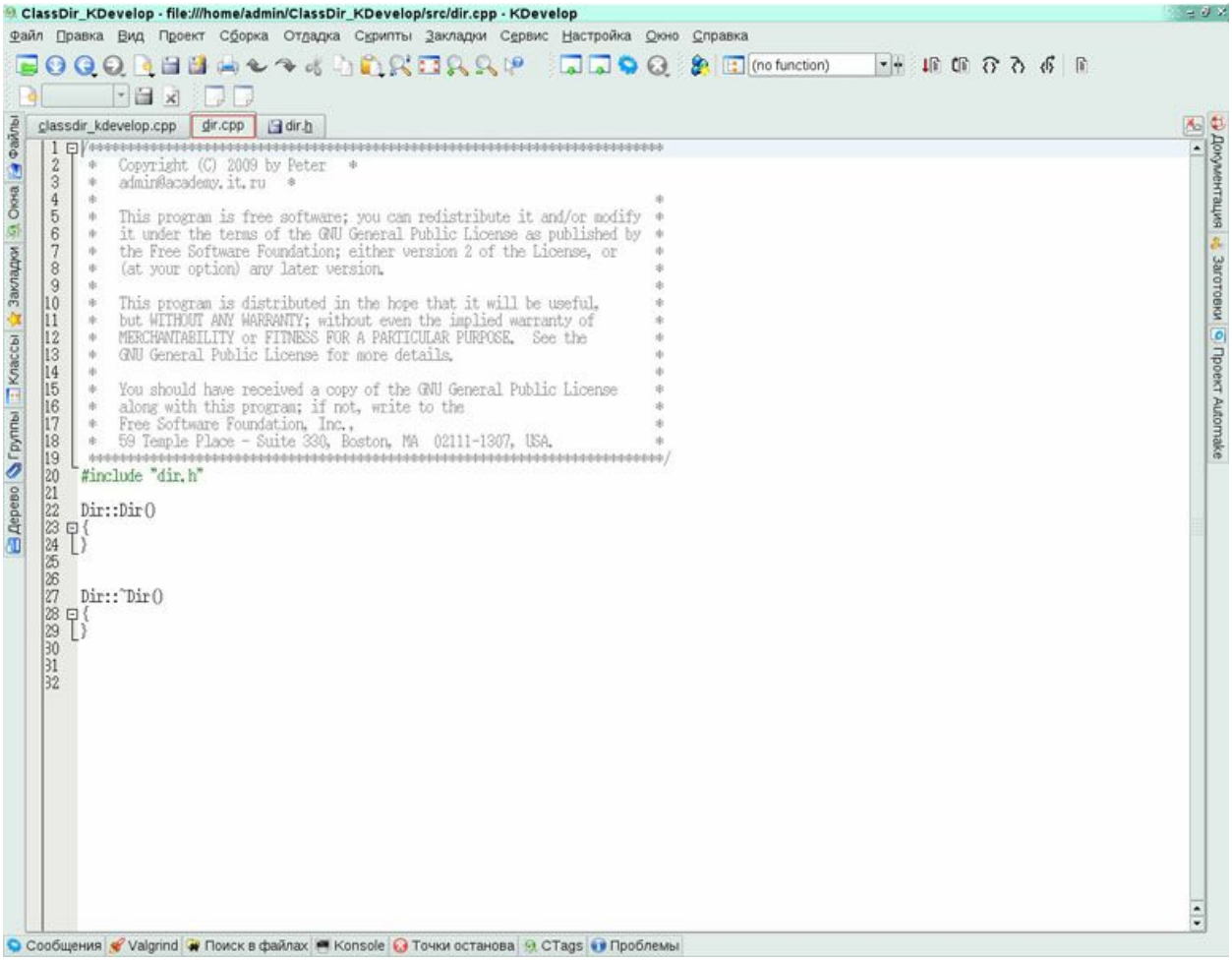

Рис. 14

Снова с целью возможного «отката» запомним стартовый текст:

```
/*************************************************************************
****
        *   Copyright (C) 2009 by Peter   *
        *   admin@academy.it.ru   *
        *
*
        *   This program is free software; you can redistribute it and/or
modify  *
        *   it under the terms of the GNU General Public License as published
by  *
        *   the Free Software Foundation; either version 2 of the License, or
*
        *   (at your option) any later version.
*
        *
*
        *   This program is distributed in the hope that it will be useful,
*
        *   but WITHOUT ANY WARRANTY; without even the implied warranty of
*
        *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
*
        *   GNU General Public License for more details.
*
        *
*
```

```
       *   You should have received a copy of the GNU General Public License
*
       *   along with this program; if not, write to the
*
       *   Free Software Foundation, Inc.,
*
       *   59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.
*
*************************************************************************/
      #include "dir.h"

      Dir::Dir()
      {
      }


      Dir::~Dir()
      {
      }
```

Здесь обращает на себя внимание применение оператора разрешения области видимости (scope resolution operator) в виде двойного двоеточия (**::**). Запись SomeClass::SomeMethod говорит о том, что мы готовы приступить к реализации метода SomeMethod принадлежащего именно классу SomeClass. Ведь классы SomeClass2, MyClass да и наш создаваемый класс **Dir** все могли бы иметь одноименные методы и нам (да и компилятору) нужно было бы как-то различать - где чья реализация. Обсуждаемый оператор данный вопрос разрешает совершенно однозначно.

18. Полностью замените текущую реализацию конструктора (строки 22-24) такой:

```
      Dir::Dir(string name,int numOfChild,bool isOwner)
      {
            _name=name;
            _numOfChild=numOfChild;
            _isOwner=isOwner;
      }
```

19. Сразу вслед за закрывающей скобкой тела конструктора (строка 28) реализуйте второй метод:

```
      void Dir::GetInfo()
      {
            cout << endl;
            cout << "Каталог " << _name << " имеет " << _numOfChild << "
файлов и под-каталогов и ";
            if(_isOwner)
                  cout << "принадлежит текущему пользователю.";
            else
                  cout << "НЕ принадлежит текущему пользователю.";
            cout << endl;
      }
```

20. Деструктор нашего класса реализации не требует, а поэтому создание класса завершенно. Финальный текст файла **dir.cpp** будет таким:

```
      /*************************************************************************
****
       *   Copyright (C) 2009 by Peter   *
       *   admin@academy.it.ru   *
       *
*
       *   This program is free software; you can redistribute it and/or
modify  *
       *   it under the terms of the GNU General Public License as published
by  *
       *   the Free Software Foundation; either version 2 of the License, or
*
       *   (at your option) any later version.
*
       *
*
       *   This program is distributed in the hope that it will be useful,
*
       *   but WITHOUT ANY WARRANTY; without even the implied warranty of
*
       *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
*
       *   GNU General Public License for more details.
*
       *
*
       *   You should have received a copy of the GNU General Public License
*
       *   along with this program; if not, write to the
*
       *   Free Software Foundation, Inc.,
*
       *   59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.
*
*************************************************************************/
      #include "dir.h"

      Dir::Dir(string name,int numOfChild,bool isOwner)
      {
            _name=name;
            _numOfChild=numOfChild;
            _isOwner=isOwner;
      }
      void Dir::GetInfo()
      {
            cout << endl;
            cout << "Каталог " << _name << " имеет " << _numOfChild << "
файлов и под-каталогов и ";
            if(_isOwner)
                  cout << "принадлежит текущему пользователю.";
            else
                  cout << "НЕ принадлежит текущему пользователю.";
            cout << endl;
      }


      Dir::~Dir()
      {
      }
```

Визуальное же его представление в окне редактора приведено на Рис. 15

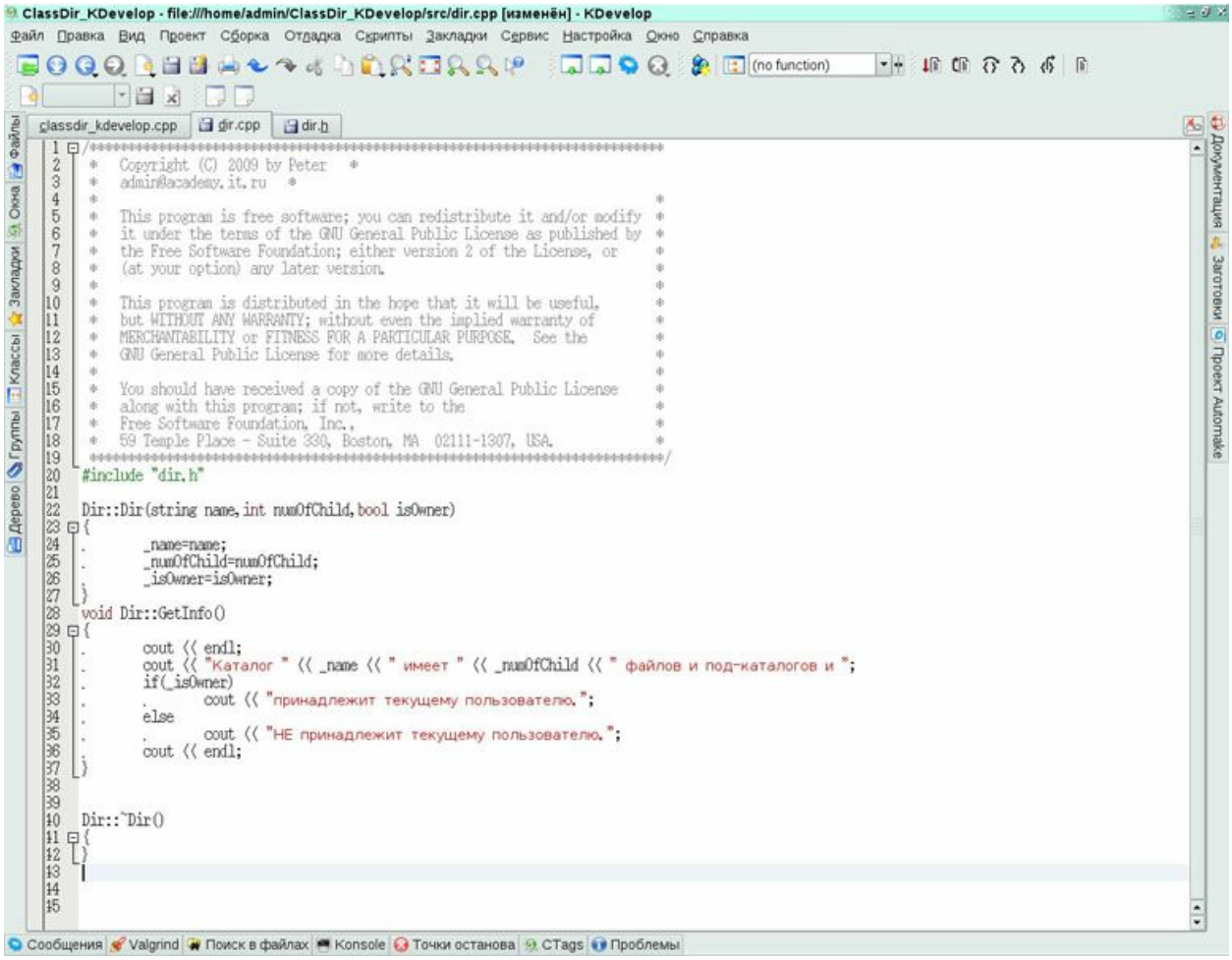

Рис. 15

21. Переключаемся на самый первый файл, classdir_kdevelop.cpp, щелчком по его закладке и приступаем к реализации метода Main. В данный момент среда сгенерировала нам код, выводящий на консоль строку «Hello, world!» и внешний вид этого файла приведен на Рис. 16.

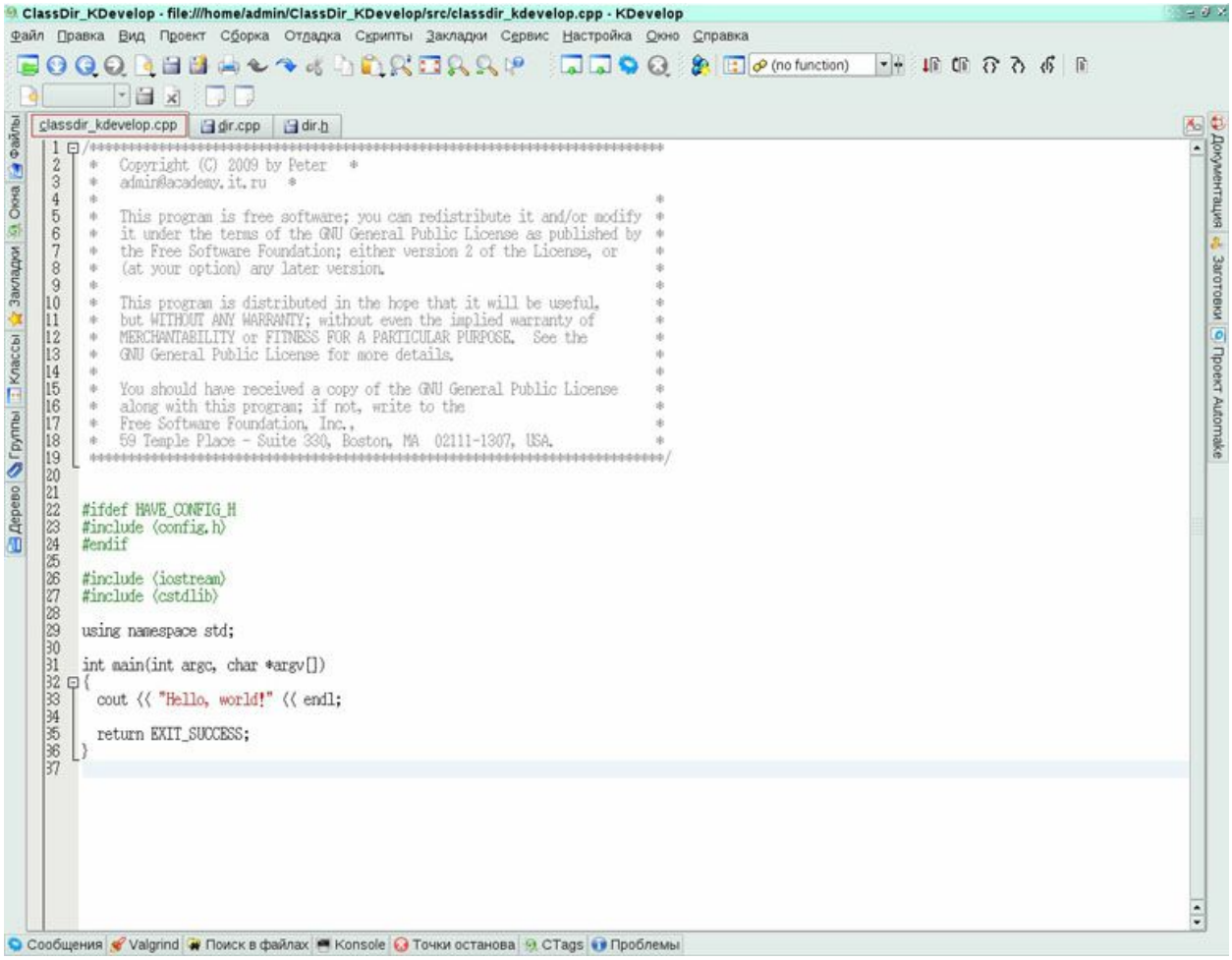

Рис. 16.

Снова с целью возможного «отката» запомним стартовый текст:

```
/*************************************************************************
****
        *   Copyright (C) 2009 by Peter   *
        *   admin@academy.it.ru   *
        *
*
        *   This program is free software; you can redistribute it and/or
modify  *
        *   it under the terms of the GNU General Public License as published
by  *
        *   the Free Software Foundation; either version 2 of the License, or
*
        *   (at your option) any later version.
*
        *
*
        *   This program is distributed in the hope that it will be useful,
*
        *   but WITHOUT ANY WARRANTY; without even the implied warranty of
*
        *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
*
        *   GNU General Public License for more details.
*
        *
*
```

```
 *   You should have received a copy of the GNU General Public License
*
 *   along with this program; if not, write to the
*
 *   Free Software Foundation, Inc.,
*
 *   59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.
*
*************************************************************************/


#ifdef HAVE_CONFIG_H
#include <config.h>
#endif

#include <iostream>
#include <cstdlib>

using namespace std;

int main(int argc, char *argv[])
{
  cout << "Hello, world!" << endl;

  return EXIT_SUCCESS;
}
```

22. Это определенно не то что нам требуется. А поэтому полностью перепишем реализацию метода Main, включая и заголовок. Сотрем строки с 31-й до конца файла и объявим на этом же месте метод Main так:

```
int main()
{

}
```

23. Перед тем как наполнить тело метода операторами вспомним, что мы планировали воспользоваться строковой константой. Объявим ее ДО метода Main, к примеру в строке 30:

```
const string YES_ANSWER("yes");
```

24. Начнем реализацию метода с декларации в строке 33 четырех локальных переменных:

```
string _name;
int _numOfChild;
string _isOwnerAsString;
bool _isOwner;
```

25. В строках 37-42 запросите у пользователя всю информацию нужную нам для создания экземпляра класса **Dir**:

```
cout << "Введите имя каталога: ";
cin >> _name;
cout << "Введите число содержащихся в нем файлов и под-каталогов: ";
cin >> _numOfChild;
```

```
      cout << "Введите " << YES_ANSWER << " если текущий пользователь владеет
каталогом, иначе нажмите Enter: ";
      cin >> _isOwnerAsString;
```

26. В строке 43 проанализируйте пользовательский ввод для последней переменной (_isOwnerAsString) и на этом основании присвойте локальной переменной **_isOwner true или false**:

```
      _isOwner=(_isOwnerAsString==YES_ANSWER);
```

27. В строке 44 задекларируйте пятую локальную переменную по имени _myDir типа **Dir**. Инициализируйте ее вызовом конструктора. Передайте последнему в качестве аргументов три соответствующих локальных переменных:

```
      Dir _myDir(_name,_numOfChild,_isOwner);
```

28. Наконец в строке 45 вызовете на новом экземпляре метод которые заставит этот экземпляр сообщить свое внутренне состояние:

```
      _myDir.GetInfo();
```

29. Убедитесь, что в строке 46 находится закрывающая фигурная скобка (**}**), завершающая тело метода Main.

30. Мы почти закончили с файлом classdir_kdevelop.cpp. Единственно, что пока не учтено, что в его тексте мы используем класс **Dir**, но метод Main "не понимает" о чем идет речь. Т.е. мы должны где-то ДО начала этого метода указать заголовочный файл используемого класса **Dir**. Строка 28 выглядит подходящей. Запишем в нее:

```
      #include "dir.h"
```

31. Финальный код файла **classdir_kdevelop.cpp** будет таким:

```
      /***************************************************************************
       *   Copyright (C) 2009 by Peter   *
       *   admin@academy.it.ru   *
       *                                                                         *
       *   This program is free software; you can redistribute it and/or modify  *
       *   it under the terms of the GNU General Public License as published by  *
       *   the Free Software Foundation; either version 2 of the License, or     *
       *   (at your option) any later version.                                   *
       *                                                                         *
       *   This program is distributed in the hope that it will be useful,       *
       *   but WITHOUT ANY WARRANTY; without even the implied warranty of        *
       *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the         *
       *   GNU General Public License for more details.                          *
       *                                                                         *
```

```
       *   You should have received a copy of the GNU General Public License
*
       *   along with this program; if not, write to the
*
       *   Free Software Foundation, Inc.,
*
       *   59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.
*
*************************************************************************/


      #ifdef HAVE_CONFIG_H
      #include <config.h>
      #endif

      #include <iostream>
      #include <cstdlib>
      #include "dir.h"
      using namespace std;
      const string YES_ANSWER("yes");
      int main()
      {
      string _name;
      int _numOfChild;
      string _isOwnerAsString;
      bool _isOwner;
      cout << "Введите имя каталога: ";
      cin >> _name;
      cout << "Введите число содержащихся в нем файлов и под-каталогов: ";
      cin >> _numOfChild;
      cout << "Введите " << YES_ANSWER << " если текущий пользователь владеет
каталогом, иначе нажмите Enter: ";
      cin >> _isOwnerAsString;
      _isOwner=(_isOwnerAsString==YES_ANSWER);
      Dir _myDir(_name,_numOfChild,_isOwner);
      _myDir.GetInfo();
      }
```

Визуальное же его представление в окне редактора приведено на Рис. 17

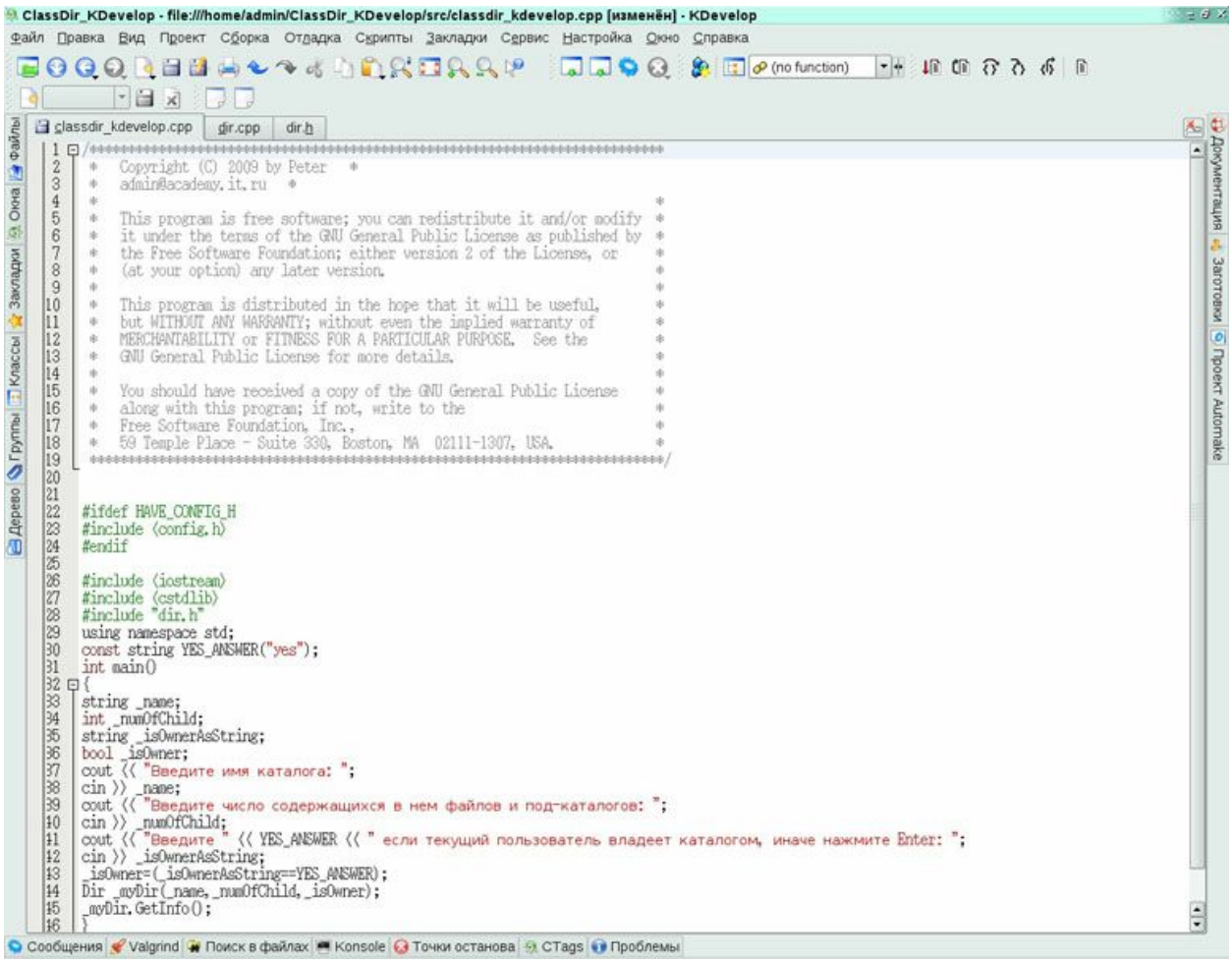

Рис. 17

32. Проект завершен, проведем его компиляцию. Для этого можно воспользоваться пунктом меню **Сборка-Собрать проект** (Рис. 18) или просто нажать **F8.**

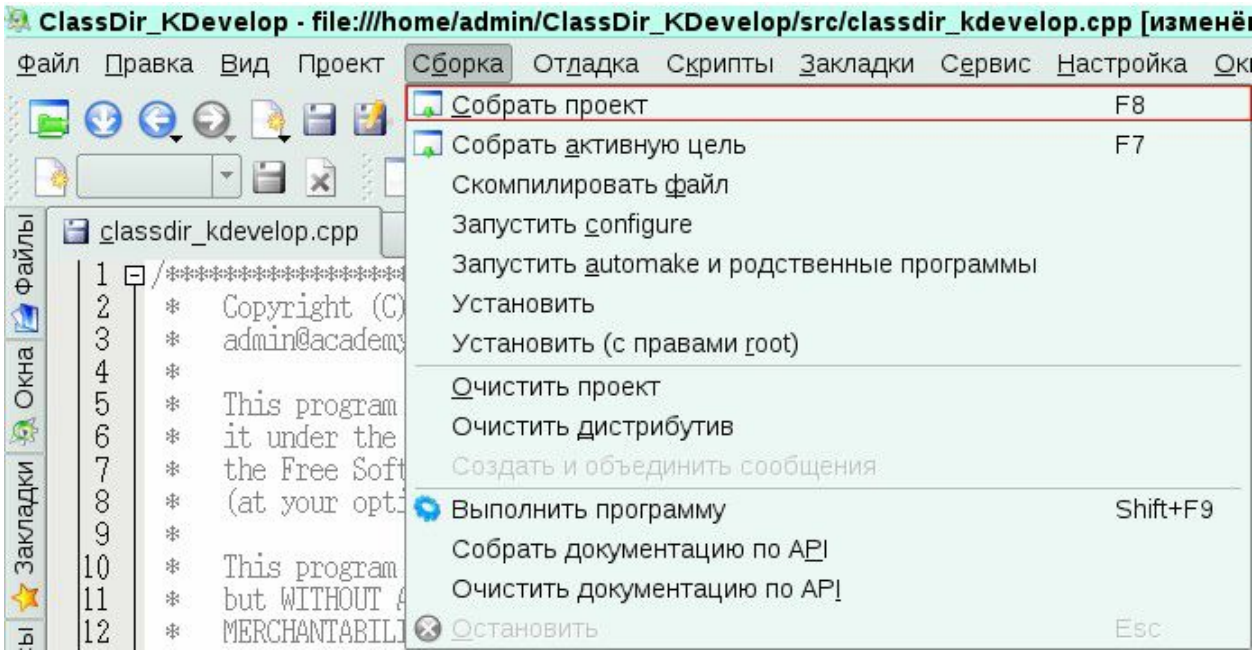

Рис. 18

33. Как пояснялось в уроке «Работа со средой разработки ПО KDevelop» в теоретической части курса при первой компиляции среде KDevelop требуется создать специальный Makefile. Среда сообщит нам об этом в окне-предупреждении (Рис. 19). Разрешите ей провести создание и настройку этого файла нажав в этом окне кнопку **Запустить.** Процесс создания этого файла весьма длителен (20-30 сек. на современном компьютере), однако проводится только при самой первой компиляции проекта. Все последующие компиляции будут значительно быстрее.

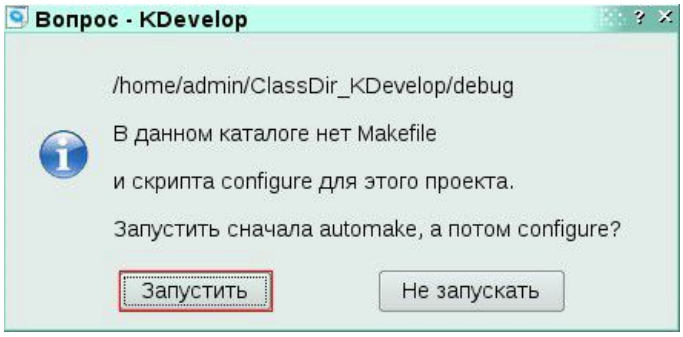

Рис. 19

34. Проанализируйте сообщения компилятора в окне Сообщения и убедитесь в отсутствии ошибок (Рис. 20). Об этом скажет **отсутствие** строк сообщений выводимых в это окно красным цветом, а так же финальное сообщение **«*** Успешное завершение ***».**

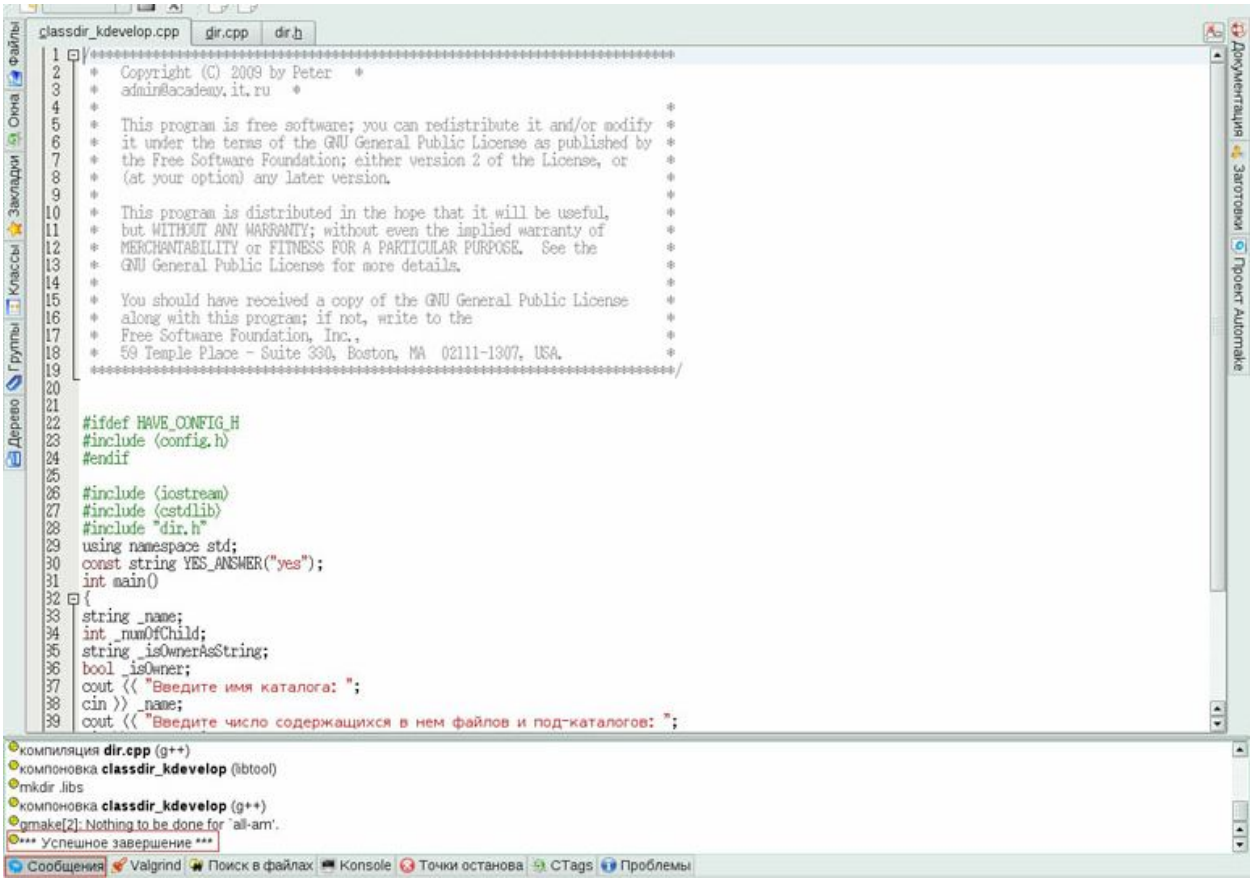

Рис. 20

35. Попробуем провести пробный сеанс работы с нашей программой. Для этого не обязательно закрывать среду разработки или переключаться на консоль. Можно выбрать пункт меню **Сборка-Выполнить программу** (Рис. 21) или просто нажать **Shift+F9**. В ответ на любое из этих действий KDevelop сам откроет консоль и запустит в ней нашу программу.

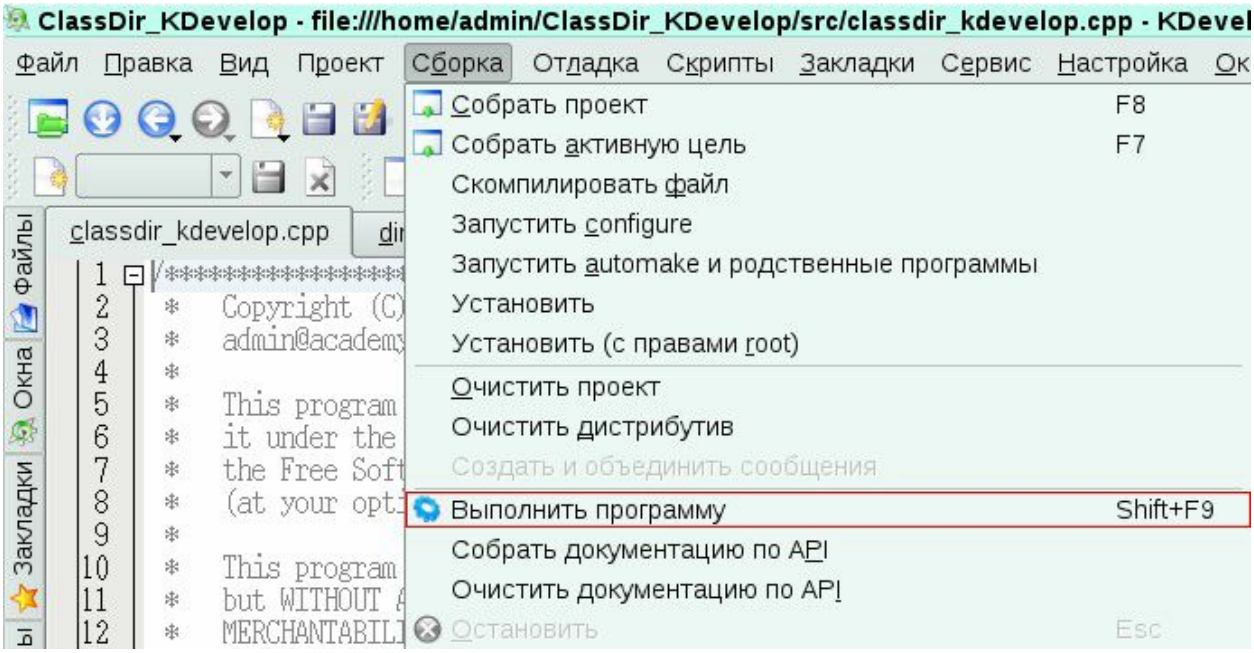

Рис. 21

36. Примите на себя роль пользователя программы **ClassDir_KDevelop** и ответьте на три вопроса подходящими сообщениями. Проанализируйте корректность вывода информации на консоль методом GetInfo() класса **Dir**. Пример диалога с нашей программой представлен на Рис. 22. В этом диалоге в качестве ответов на вопросы были использованы строки «MyDocs», «187», «yes».

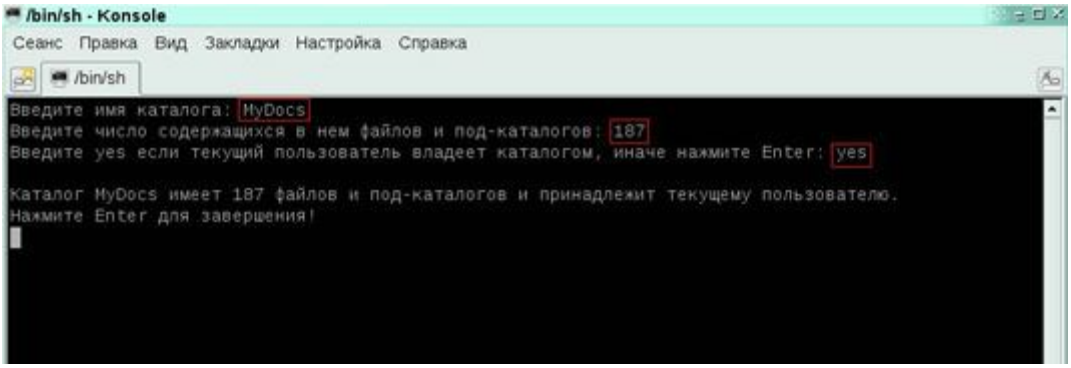

Рис. 22

37. Как предписывает нам последняя строчка в консольной сессии (Рис. 22) нажмите **Enter**, что бы завершить ее и вернуться в среду разработки KDevelop.

38. Работа над проектом окончена. Закройте KDevelop с помощью меню **Файл-Выход** (Рис. 23)

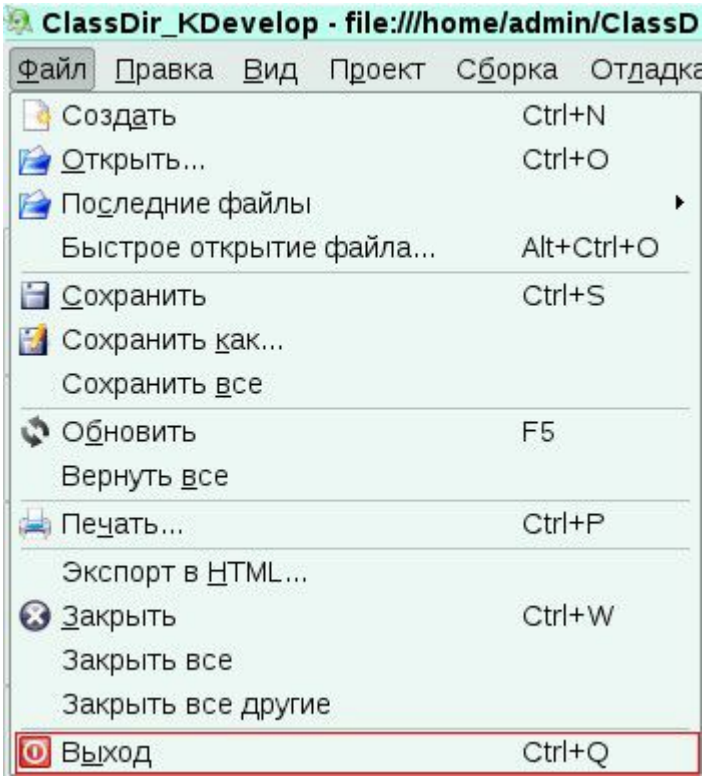

Рис. 23

39. Посмотрим, где на жестком диске располагается скомпилированный нами исполнимый модуль. Это позволит нам распространить нашу программу для всех желающих воспользоваться ее функционалом. Откройте универсальный браузер Conqueror и перейдите в наш домашний каталог (/home/admin). В списке содержащихся в нем подпапок можно видеть и корневую папку нашего проекта - **Classier_Develop**. (Рис. 24)

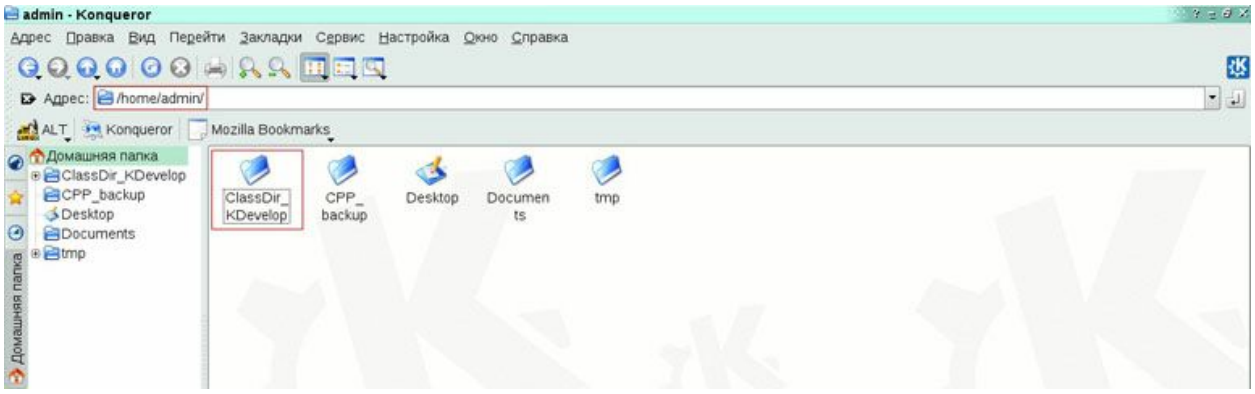

Рис. 24

40. В дереве папок слева последовательно раскройте папки **ClassDir_KDevelop** и **debug**. Выберите единственную подпапку с именем **src**. В ее содержимом справа можно видеть иконку приложения с именем *classdir_kdevelop* (Рис. 25). Это единственный файл, который мы можем/должны распространить заинтересованным лицам. Запуск его из командной строки консоли приведет к инициированию только что опробованному нами диалогу из трех вопросов и одного информационного сообщения.

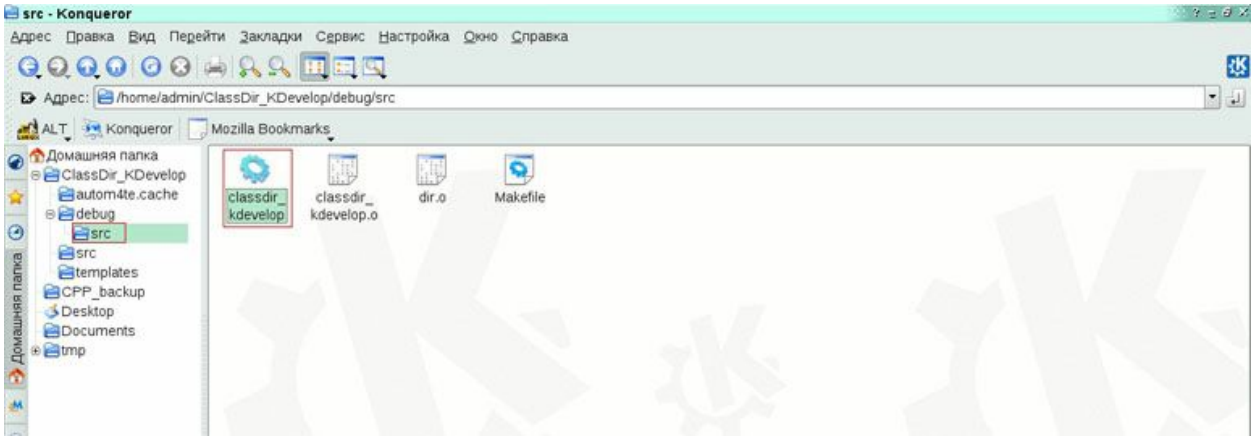

Рис. 25

2. Практика работы с Lazarus и языком программирования Pascal

1. Итак, целью этого практического занятия будет решение той же самой задачи обозначенной в разделе **«Постановка задачи»**, однако на этот раз в среде Lazarus и языке Pascal.

Убедитесь, что вы вошли в систему с именем пользователя (login) **admin**. После этого из главного меню **КДЕ-Прочие-Разработка-Lazarus** запустите среду разработки Lazarus (Рис. 26).

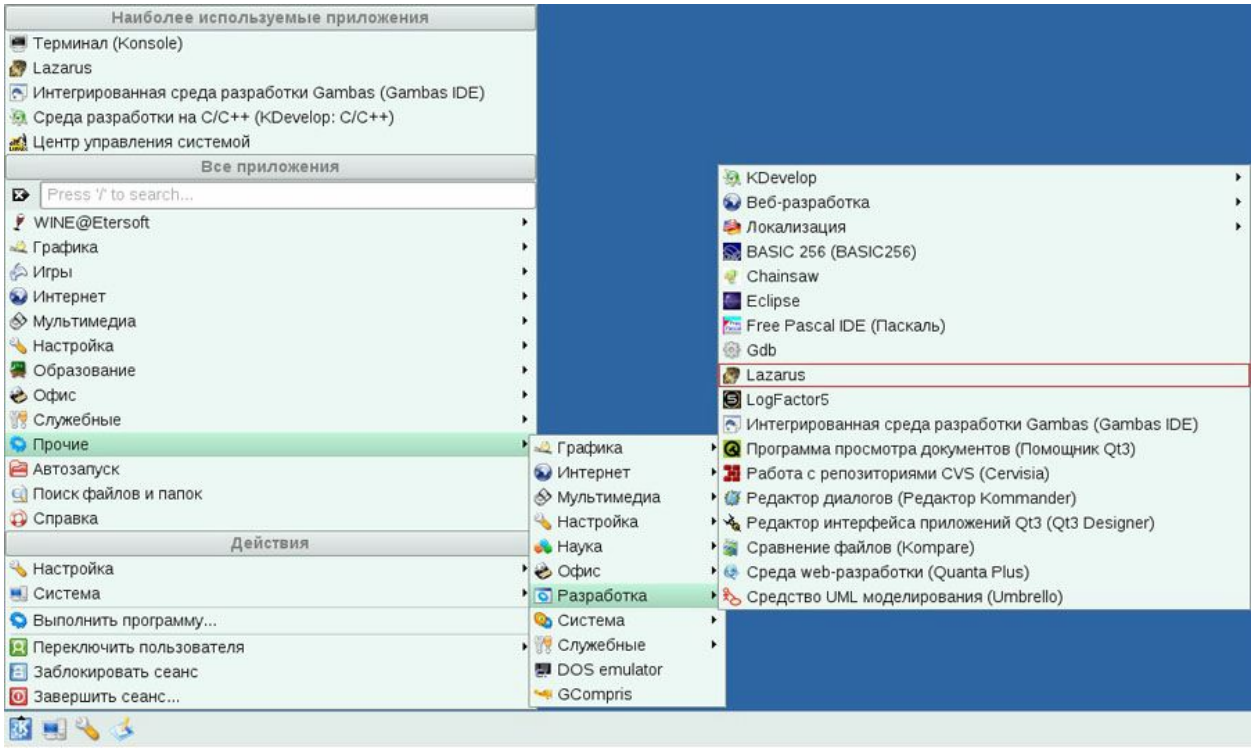

Рис. 26

2. Среда открывает проект по умолчанию, однако нам нужен проект иного типа, консольного. Поэтому текущий проект мы закроем выбрав в меню **Проект-Закрыть проект** (Рис. 27).

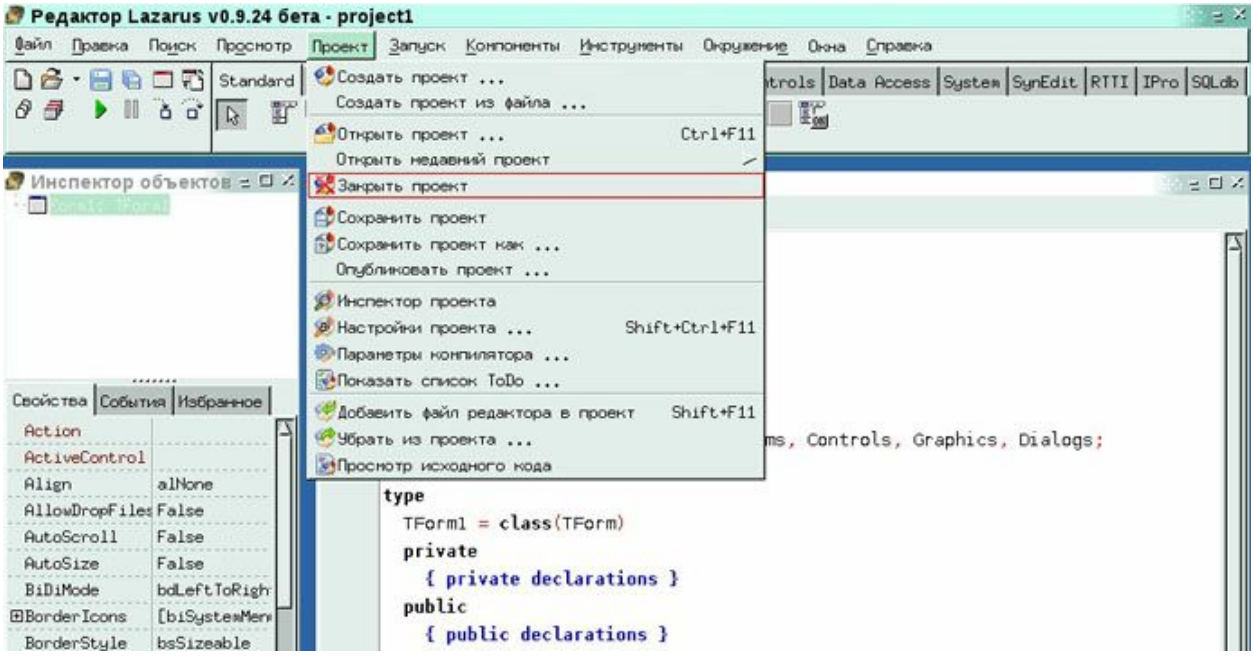

Рис. 27

3. Из трех предложенных вариантов продолжения работы укажите **«Создать новый проект»** (Рис. 28).

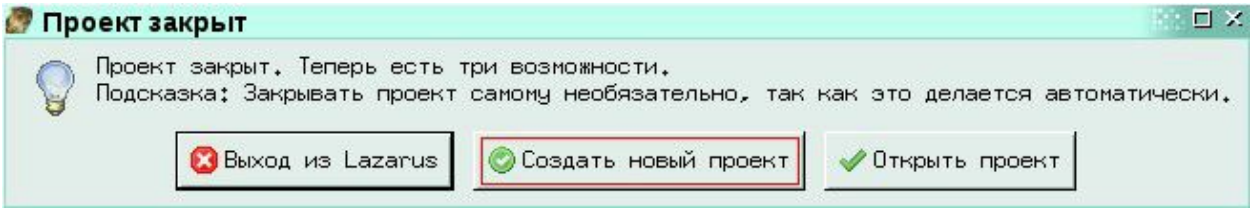

Рис. 28

4. В окне **«Создать новый проект»** выберите шаблон **«Программа пользователя»** и нажмите **Создать** (Рис. 29).

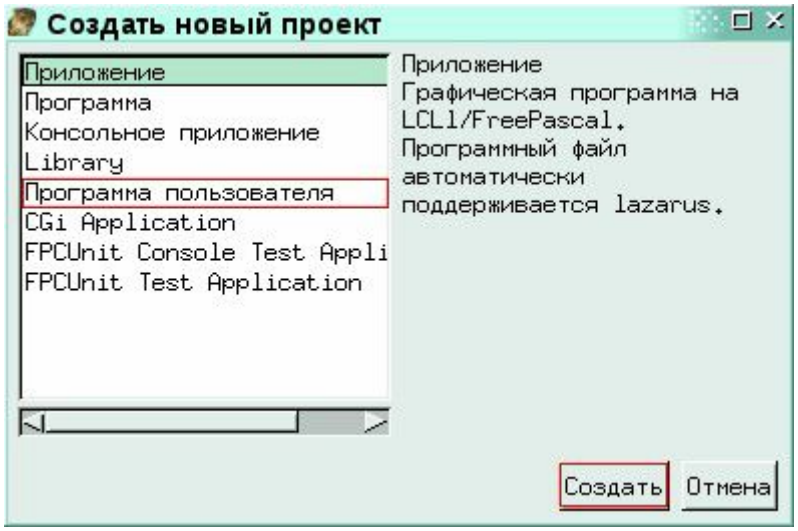

Рис. 29

5. Среда создаст первый (и пока единственный) файл нашего проекта - **project1** и откроет его в редакторе (Рис. 30). Поскольку мы собираемся полностью

переписать текст этого файла (а не отредактировать его), то нумерация строк не является жизненной необходимостью, и включать ее мы не станем. Однако, что бы иметь возможность начать снова с этого места «запомним» исходный текст стартового файла:

```
program Project1;

{$mode objfpc}{$H+}

uses
  Classes, SysUtils
  { you can add units after this };

begin
end.
```

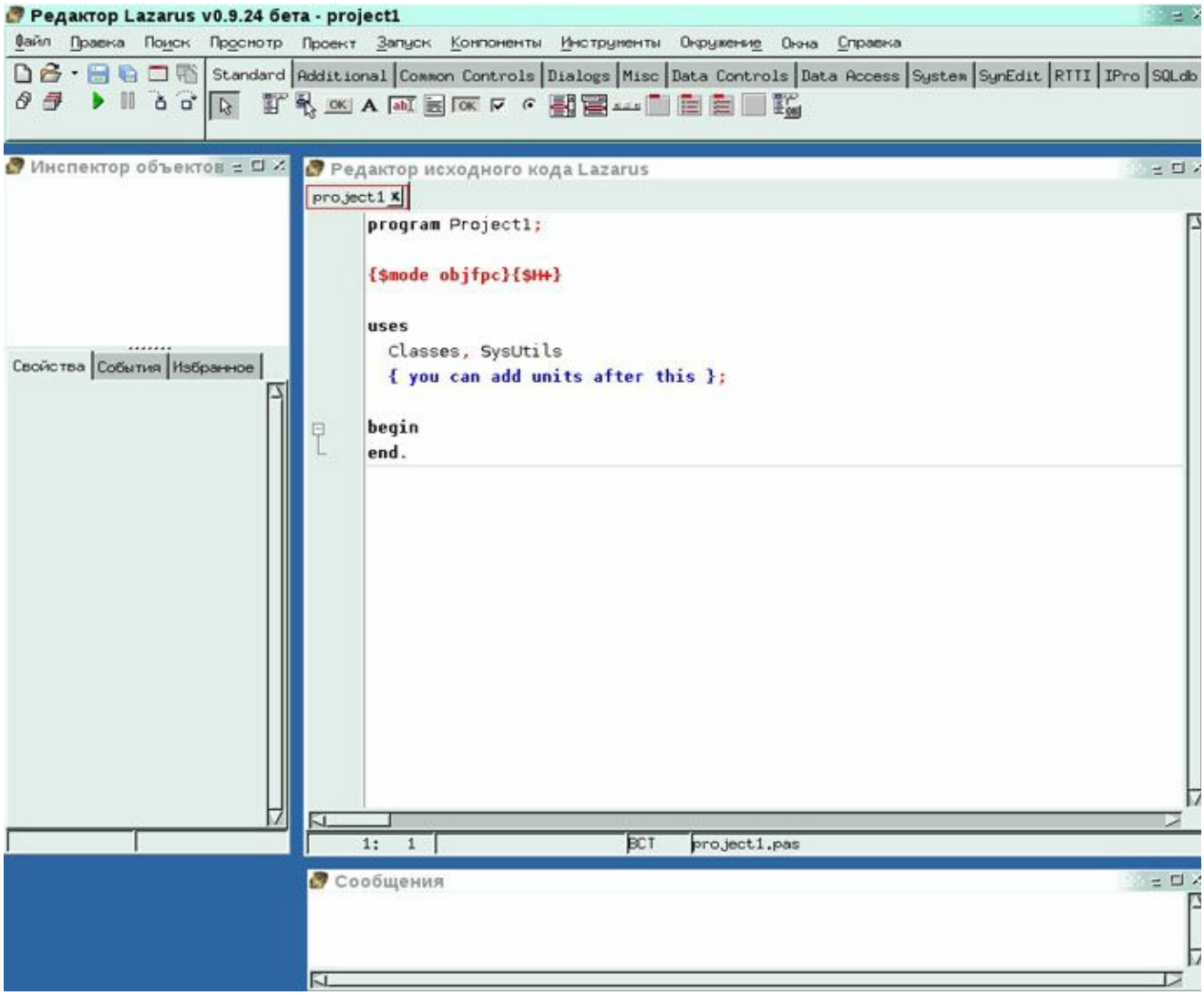

Рис. 30

6. Прежде чем приступать к редактированию сохраним файл **project1** на диск и, вместе с ним, сохраним и весь проект. Так же организуем корневую папку нашего проекта. Выберите в меню **Файл-Сохранить как….** (Рис. 31). Это должно привести в появлению диалога сохранения проекта (Рис. 32). Убедитесь, что мы находимся в нашей домашней папке, т.е. что в поле *Looking in:* указано */home/admin* (значение по умолчанию). Для создания корневой папки проекта нажмите кнопку **Создать каталог** этого диалога.

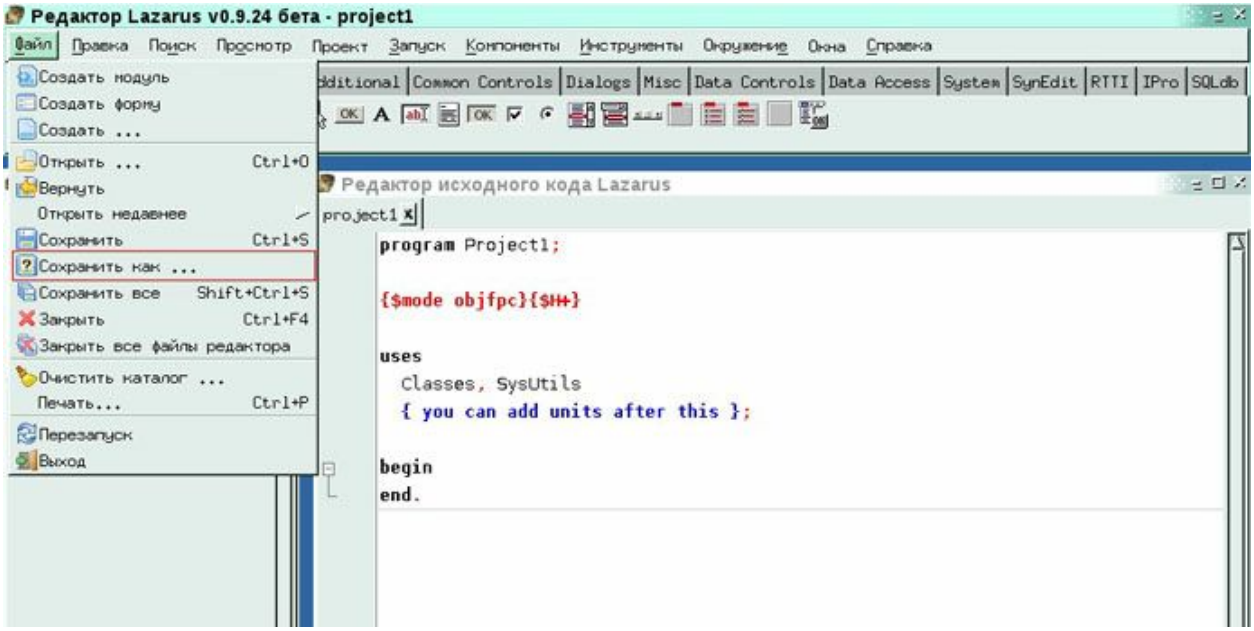

Рис. 31

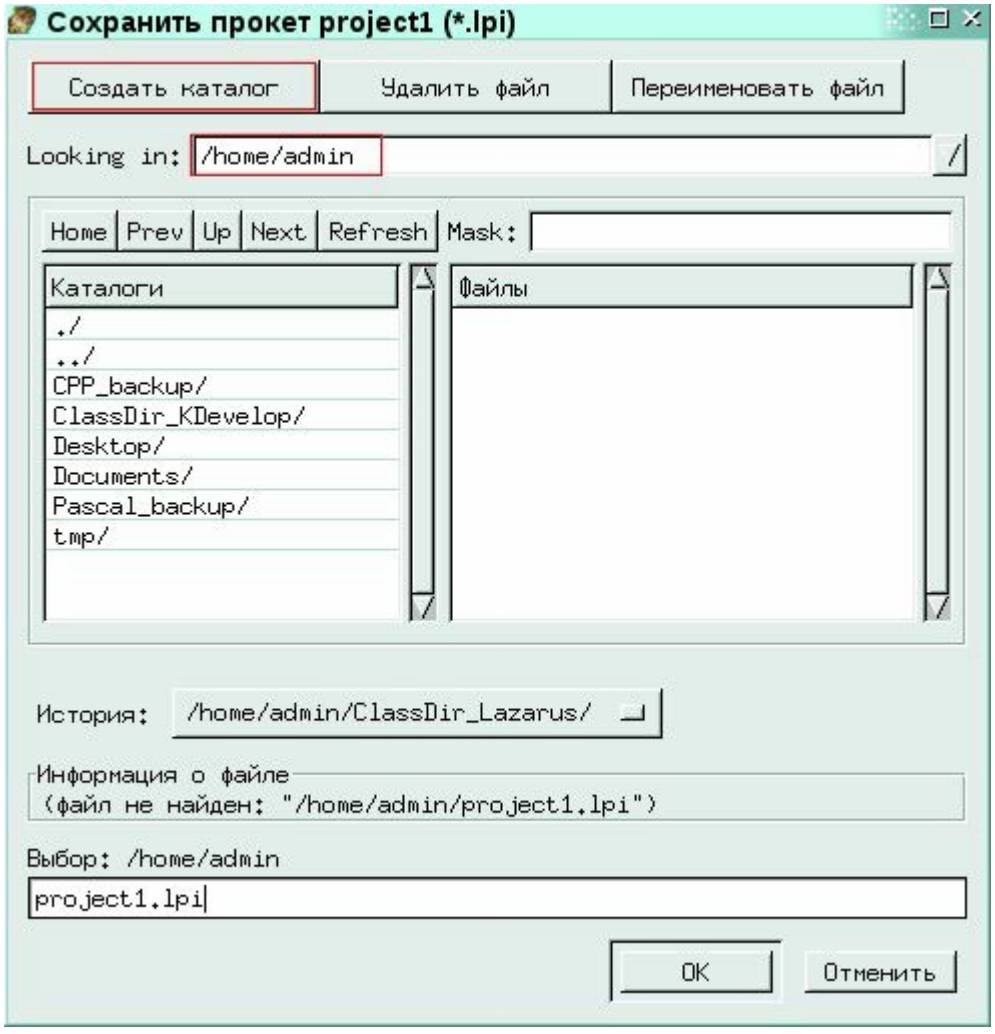

Рис. 32

7. В окне **«Создать каталог»** введите имя корневой папки нашего проекта - **ClassDir_Lazarus** - и нажмите **Создать** (Рис. 33).

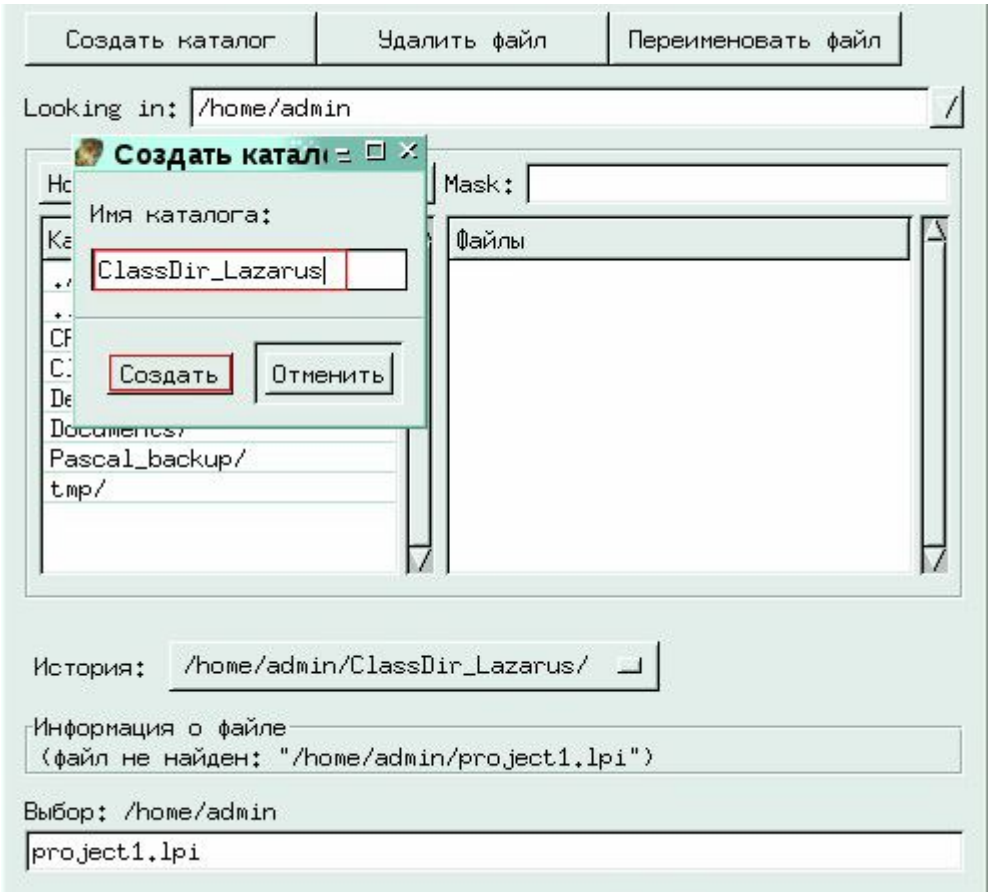

Рис. 33

8. Теперь в окне сохранения проекта, слева, в списке **Каталоги** должна появиться наша папка. Перейдите в нее двойным щелчком по ее имени (Рис. 34).

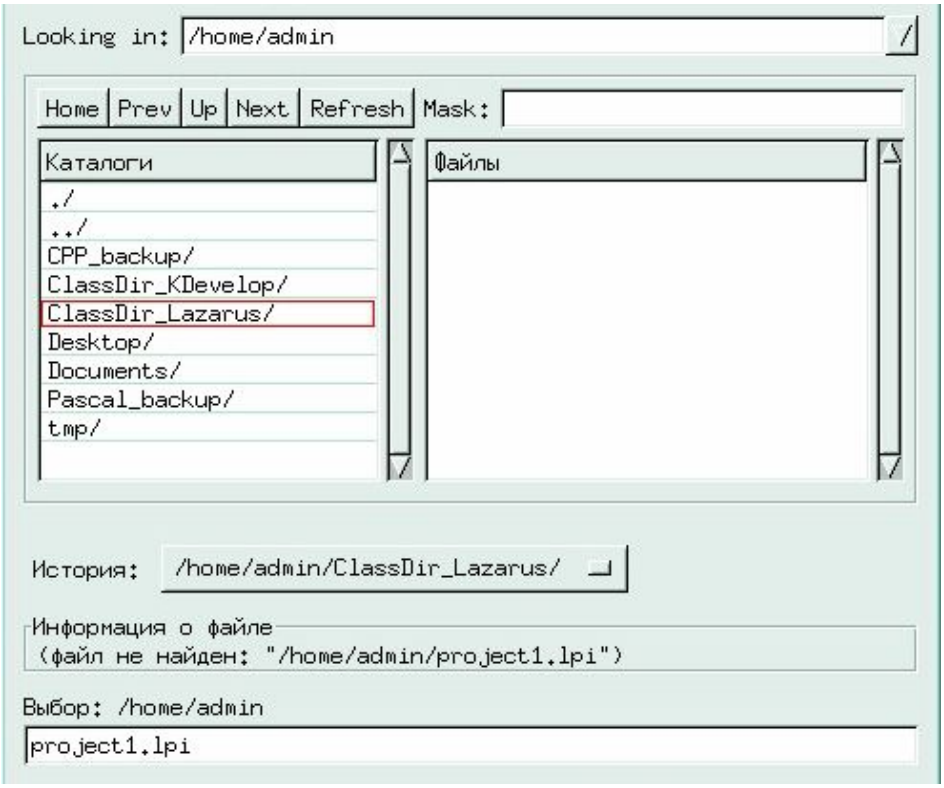

Рис. 34

9. Убедитесь, что в поле текущего местоположения (*Looking in:*) теперь указана корневая папка проекта, **/home/admin/ClassDir_Lazarus**. После чего в нижнем текстовом поле укажите имя текущего (сохраняемого) файла. Оно же станет и именем всего проекта. Остановимся на том же имени, что и корневая папка проекта - **ClassDir_Lazarus**. Проверьте еще раз правильность ввода и текущего местоположения и нажмите **OK** (Рис. 35).

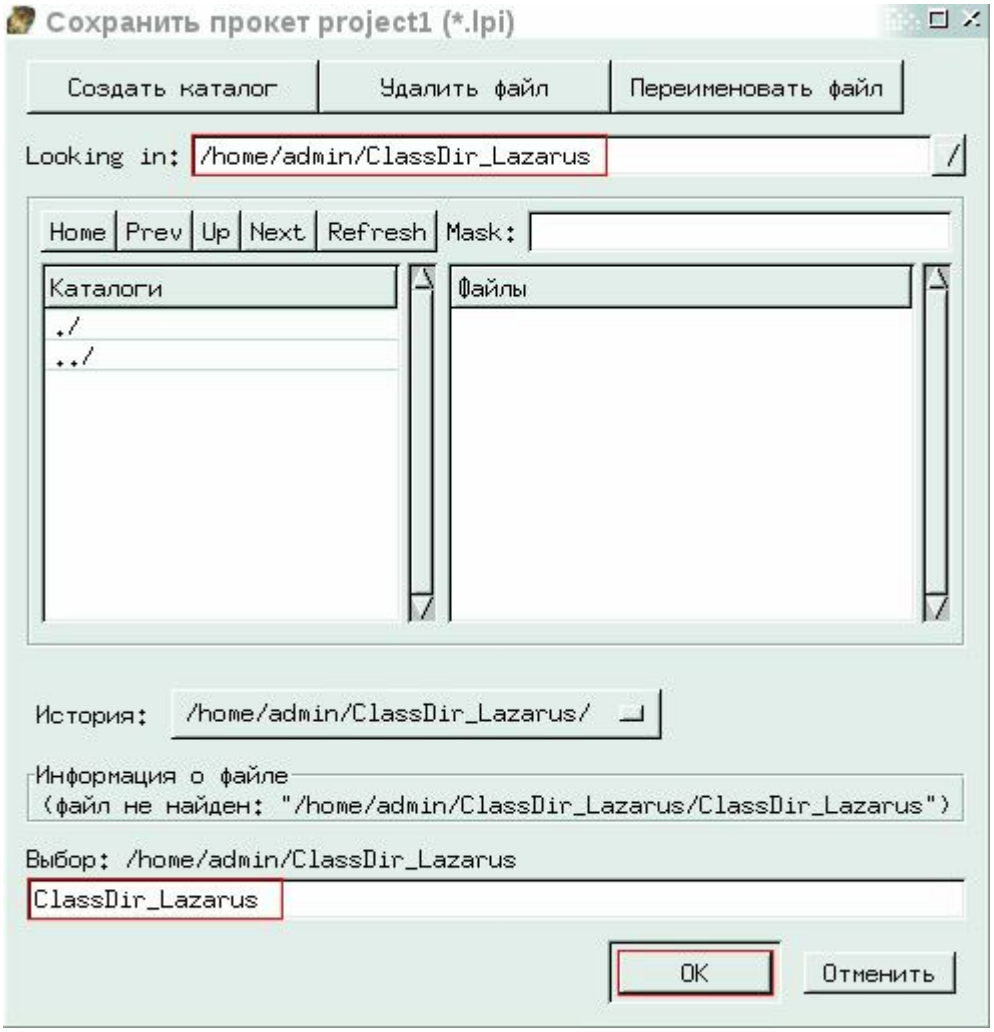

Рис. 35

10. Файл и проект успешно сохранены. Давайте приступим к написанию кода. Однако начнем не с файла главной программы открытой в настоящий момент в редакторе, а с несуществующего пока файла класса **Dir**. Добавим его в наш проект, для чего выберите пункты меню **Файл-Создать модуль** (Рис. 36).

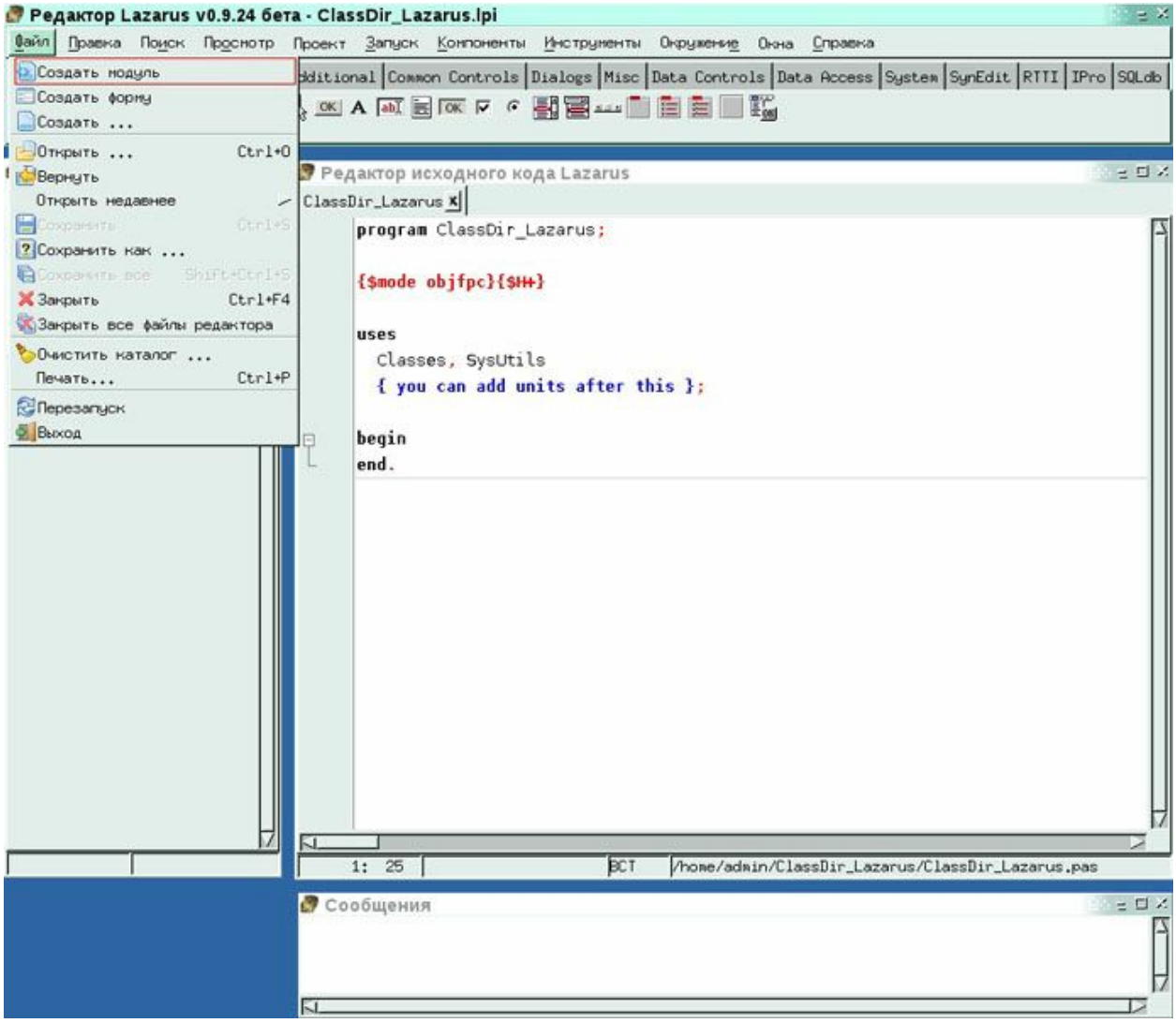

Рис. 36

11. Новый файл (модуль) добавляется к нашему проекту и открывается в редакторе. Дадим ему подходящее имя и сохраним на диск. Снова выбираем **Файл-Сохранить как….** (Рис. 31). Открывается окно практически идентичное окну сохранения проекта. Убеждаемся, что мы по-прежнему находимся в корневой папке проекта (проверяем значение поля *Looking in:*) и в нижнем текстовом поле вводим имя нового файла - **dirunit** (обратите внимание на нижний регистр всех символов!). Проверьте еще раз правильность ввода и текущего местоположения и нажмите **OK** (Рис. 37).

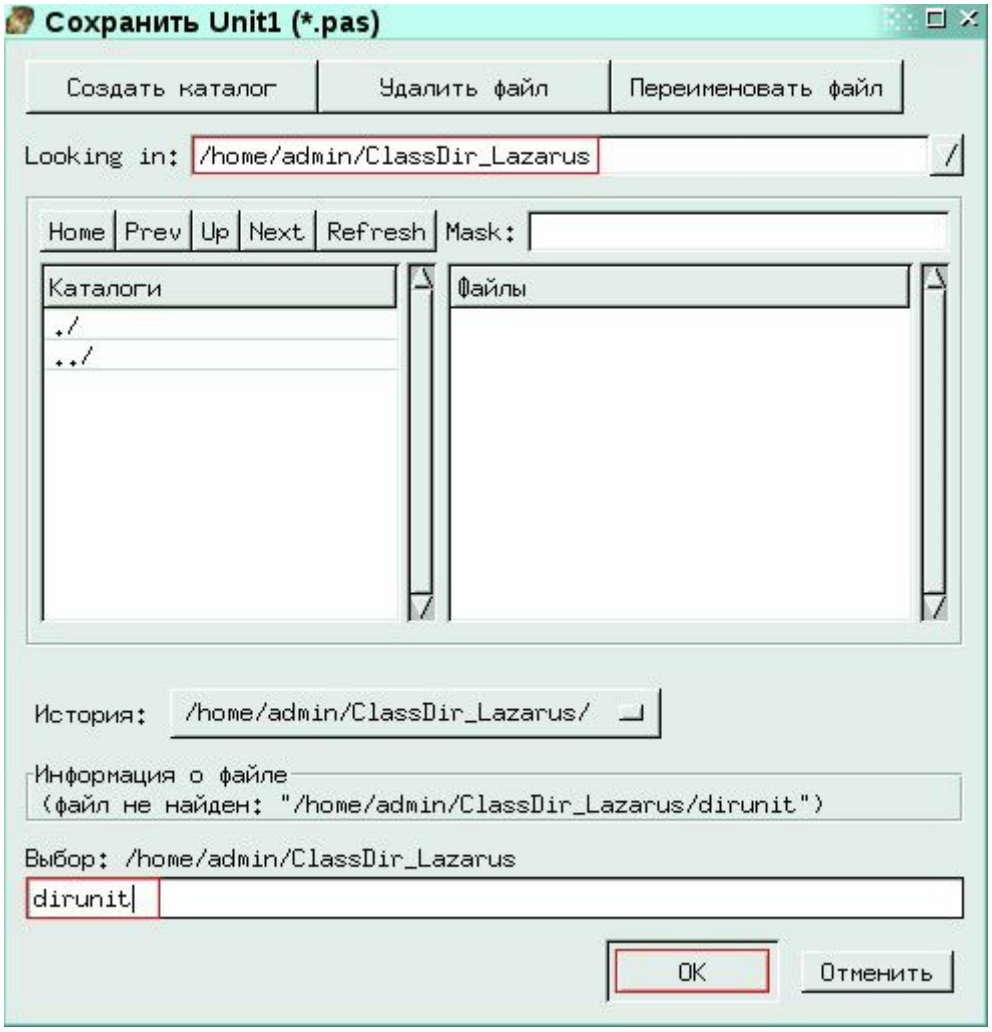

Рис. 37

12. Итак, мы готовы написать код класса **Dir** согласно дизайн-схеме нашего проекта. Для возможности начать все сначала зафиксируем текущее содержимое файла **dirunit**:

```
unit dirunit;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils;

implementation

end.
```

Убедитесь, что в редакторе активна закладка именно этого файла (Рис. 38).

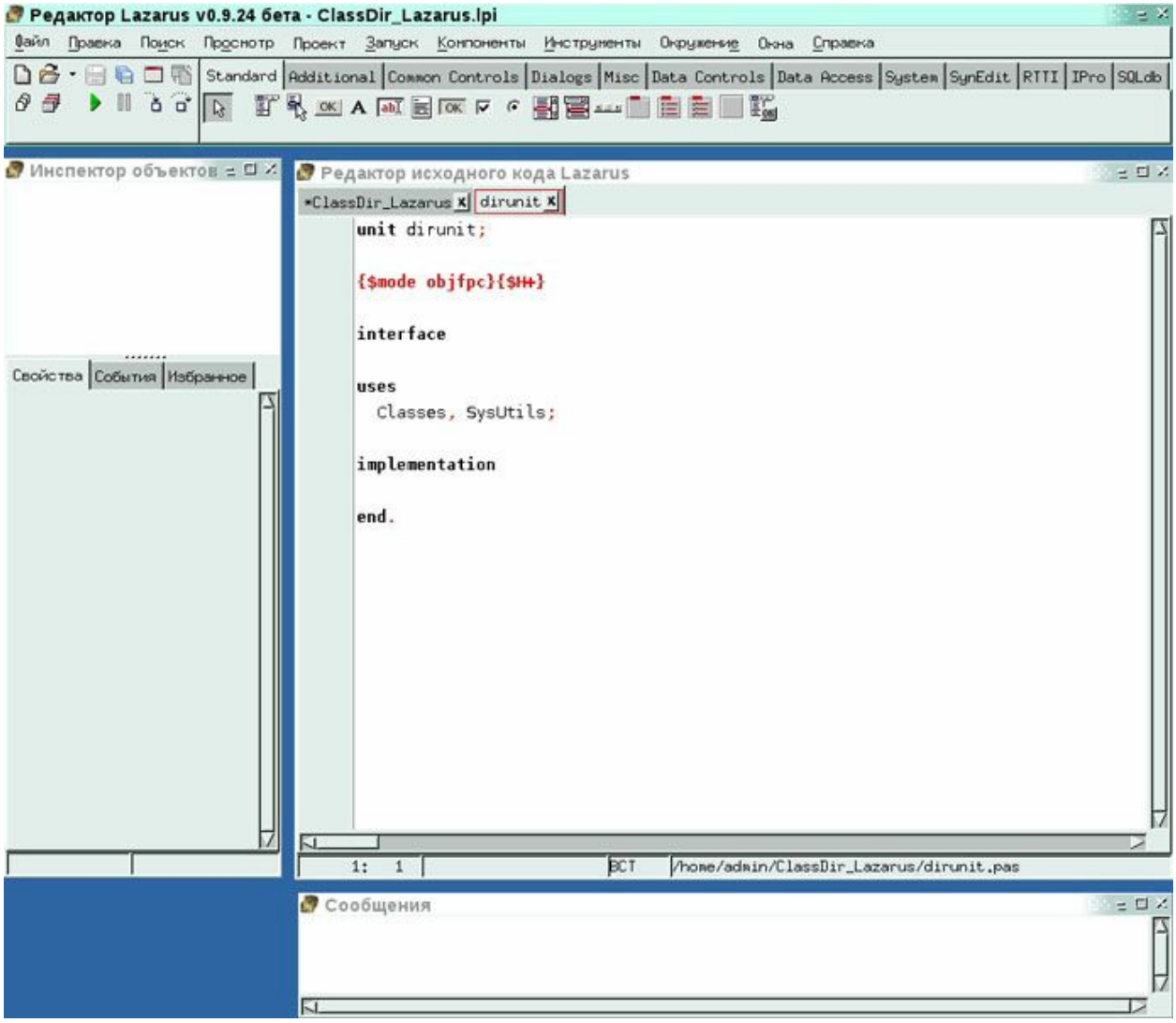

Рис. 38

13. Мы не будем пользоваться кодом из шаблона файла **dirunit**, а напишем его самостоятельно, с нуля. Поэтому полностью сотрите все текущее его содержимое и начните с декларации юнита (модуля):

```
unit DirUnit;
```

14. Откройте в следующей строке декларативную часть юнита ключевым словом **interface**:

```
interface
```

15. В этой части нам требуется задекларировать сам класс **Dir**, все его поля, конструктор и метод GetInfo. Однако реализацию обоих методов мы откладываем до второй части юнита. Ваш код должен напоминать следующий:

```
Type
   Dir = Object
private
      _name : string;
      _numOfChild : integer;
```

```
            _isOwner : boolean;
      public
            Constructor Init(name : string; numOfChild : integer;
isOwner:boolean);
            procedure GetInfo;
```

16. Завершите декларативную часть юнита ключевым словом **end**:

```
      end;
```

17. Откройте вторую, реализационную часть юнита ключевым словом **implementation**:

```
      implementation
```

18. В этой части нам требуется написать код обоих методов класса **Dir**. Начните с его конструктора:

```
       Constructor   Dir.Init(name   :   string;   numOfChild   :   integer;
isOwner:boolean);
       begin
      _name := name;
            _numOfChild := numOfChild;
            _isOwner := isOwner;
       end;
```

Обратите внимание, что перед именем метода (конструктора, в данном случае) должно идти имя класса которому этот метод и принадлежит.

19. Реализуйте процедуру **GetInfo**:

```
      procedure Dir.GetInfo;
      begin
           writeln;
           write('Каталог ', _name, ' имеет ', _numOfChild, ' файлов и под-
каталогов и ');
           if (_isOwner) then
                 write('принадлежит текущему пользователю.')
           else
                 write('НЕ принадлежит текущему пользователю.');
           writeln;
      end;
```

20. Завершите код модуля **dirunit** ключевым словом **end** (обратите внимание на точку в конце):

```
      end.
```

21. Итоговый код модуля **dirunit** будет примерно таким:

```
      unit DirUnit;
      interface
      Type
```

```
         Dir = Object
      private
            _name : string;
            _numOfChild : integer;
            _isOwner : boolean;
      public
            Constructor Init(name : string; numOfChild : integer;
isOwner:boolean);
            procedure GetInfo;
      end;
      implementation
        Constructor Dir.Init(name : string; numOfChild : integer;
isOwner:boolean);
        begin
            _name := name;
            _numOfChild := numOfChild;
             _isOwner := isOwner;
        end;
        procedure Dir.GetInfo;
        begin
             writeln;
             write('Каталог ', _name, ' имеет ', _numOfChild, ' файлов и под-
каталогов и ');
             if (_isOwner) then
                    write('принадлежит текущему пользователю.')
             else
                    write('НЕ принадлежит текущему пользователю.');
             writeln;
        end;
      end.
```

А визуально, в редакторе, он будет выглядеть как на Рис. 39, 40:

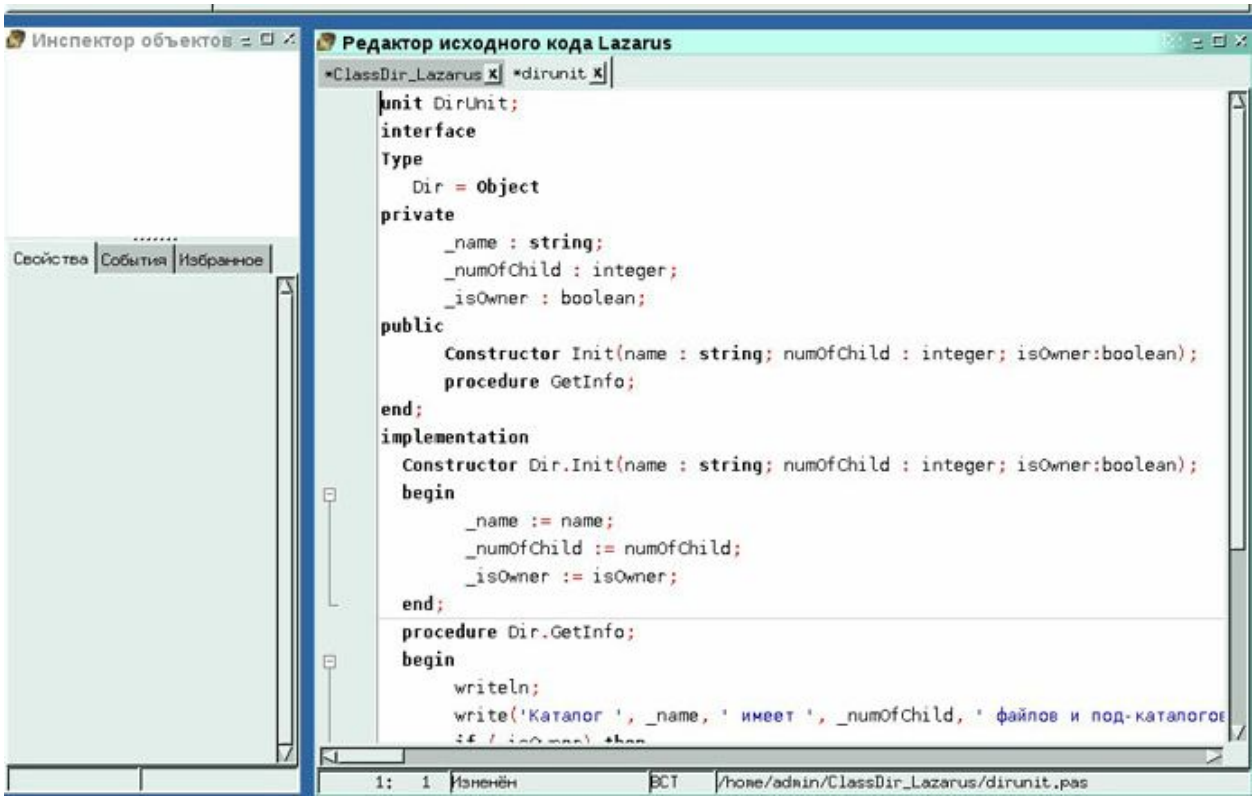

Рис. 39

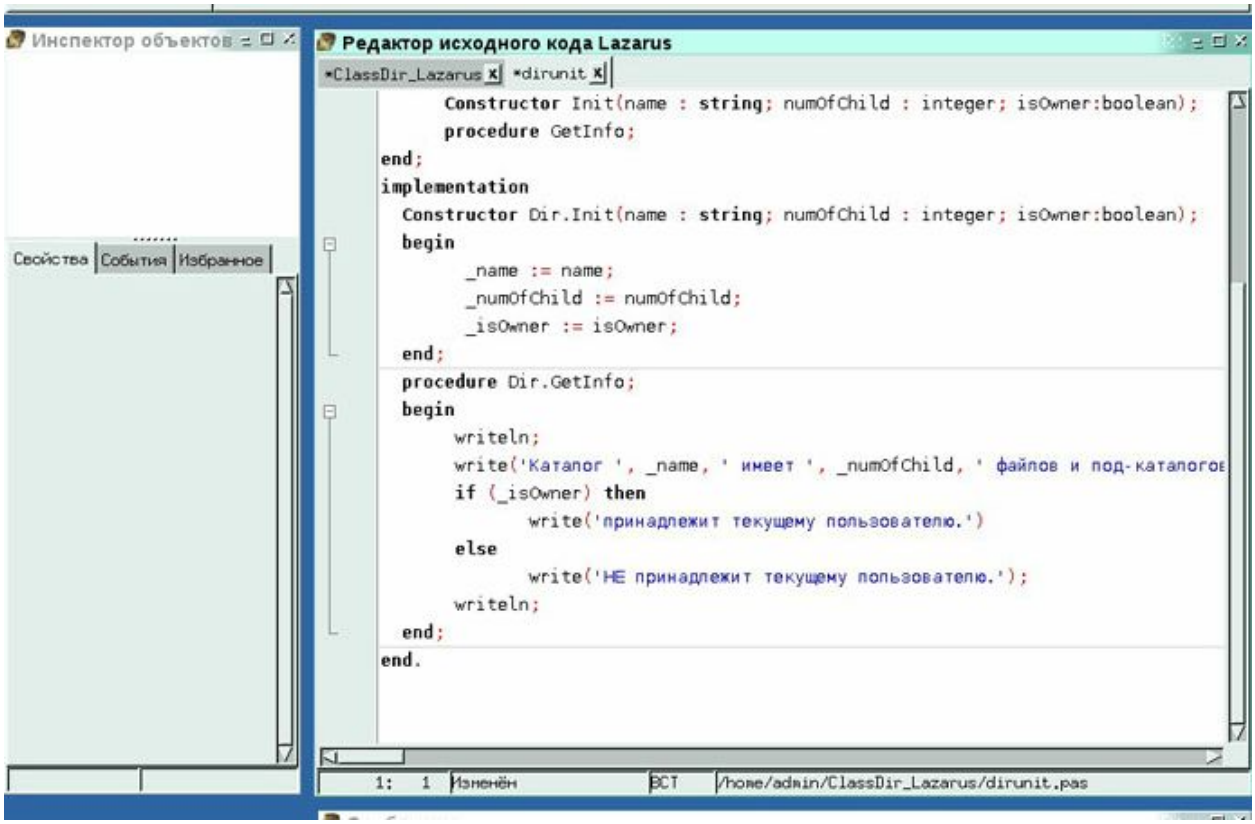

Рис. 40

22. Сохраните проделанную работу на диск для чего выберите пункты меню **Файл-Сохранить** (Рис. 41).

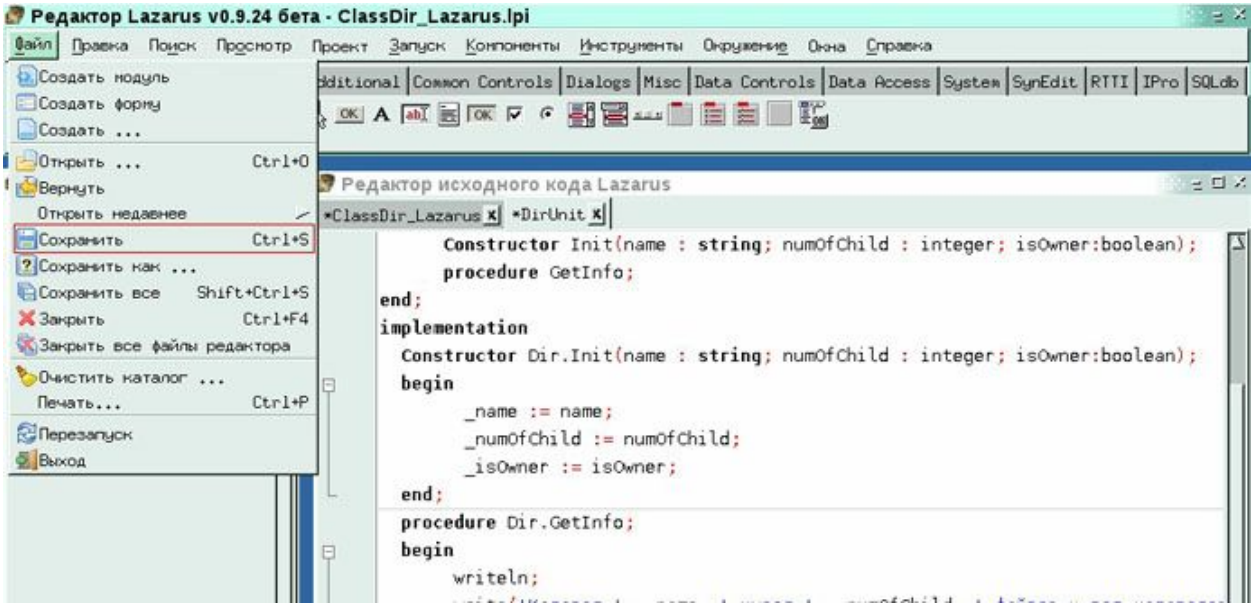

Рис. 41

23. Переключитесь на файл кода главной программы, для чего щелкните по ярлыку **ClassDir_Lazarus** в заголовке редактора (Рис. 42).

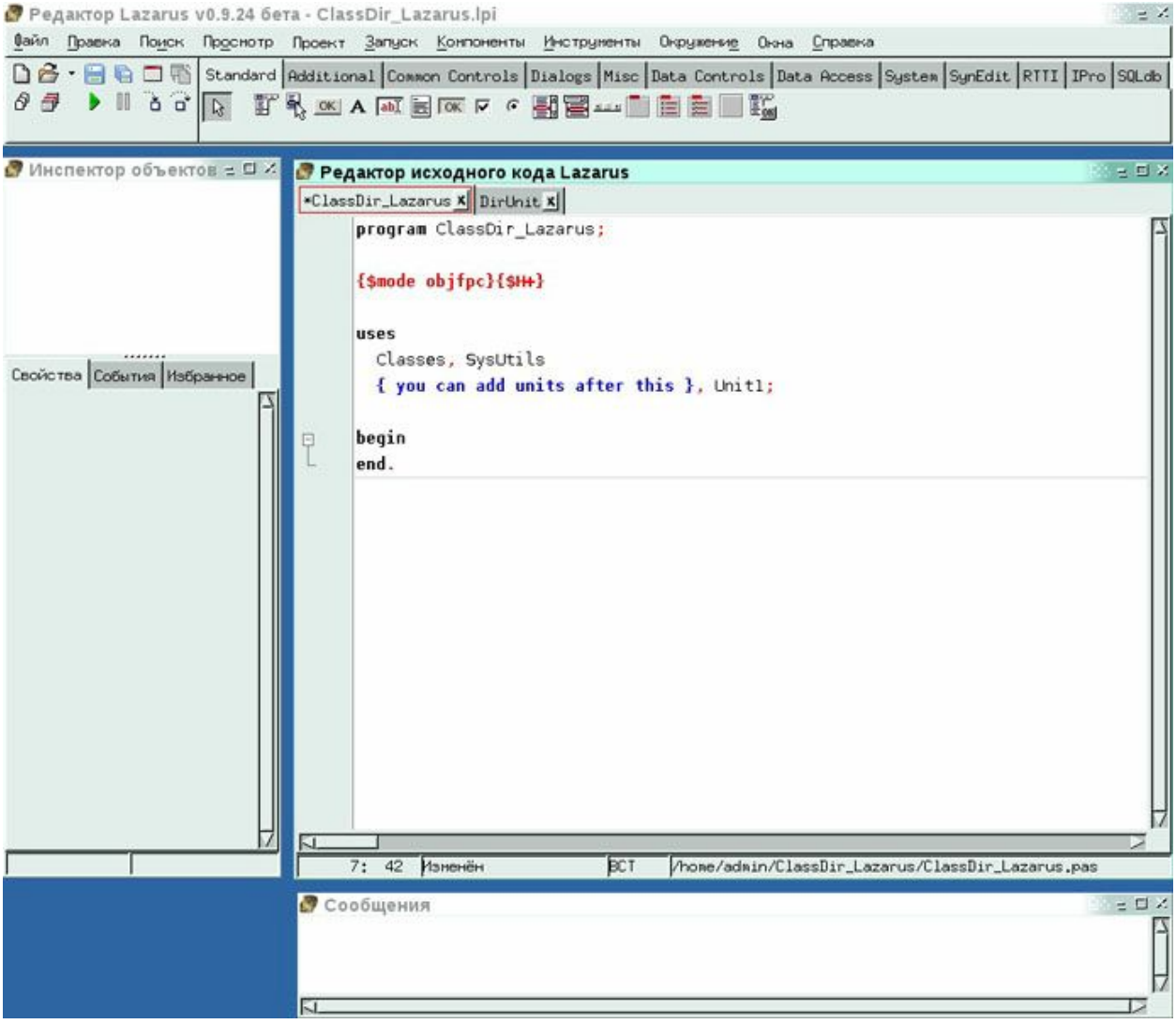

Рис. 42

24. Как и с файлом класса **Dir** напишем код главной программы с нуля. Поэтому снова полностью сотрите все текущее содержимое активного окна редактора и начните с заголовка программы:

```
program ClassDir_Lazarus;
```

25. В секции **uses** объявите, что наша главная программа будет использовать тип декларированный и реализованный в юните **DirUnit**:

```
uses DirUnit;
```

26. В секции **const** объявите требуемую нам по условию задачи строковую константу:

```
const YES_ANSWER='yes';
```

27. В секции var объявите все 5 переменных, которыми будет оперировать наша программа:

```
var
   _name, _isOwnerAsString : string;
   _numOfChild : integer;
   _isOwner : boolean;
   _myDir : Dir;
```

28. В операторных скобках **begin…end** реализуйте тело главной программы. Не забудьте о точке после закрывающей скобки (end):

```
begin
     write('Введите имя каталога: ');
     readln(_name);
     write('Введите число содержащихся в нем файлов и под-каталогов: ');
     readln(_numOfChild);
     write('Введите ', YES_ANSWER, ' если текущий пользователь владеет
каталогом, иначе нажмите Enter: ');
     readln(_isOwnerAsString);
     _isOwner:=(_isOwnerAsString=YES_ANSWER);
     _myDir.Init(_name, _numOfChild, _isOwner);
     _myDir.GetInfo;
end.
```

29. Итоговый код модуля **ClassDir_Lazarus** будет примерно таким:

```
program ClassDir_Lazarus;
uses DirUnit;

const YES_ANSWER='yes';
var
   _name, _isOwnerAsString : string;
   _numOfChild : integer;
   _isOwner : boolean;
   _myDir : Dir;
begin
     write('Введите имя каталога: ');
     readln(_name);
     write('Введите число содержащихся в нем файлов и под-каталогов: ');
     readln(_numOfChild);
     write('Введите ', YES_ANSWER, ' если текущий пользователь владеет
каталогом, иначе нажмите Enter: ');
     readln(_isOwnerAsString);
     _isOwner:=(_isOwnerAsString=YES_ANSWER);
     _myDir.Init(_name, _numOfChild, _isOwner);
     _myDir.GetInfo;
```

```
end.
```

А визуально, в редакторе, он будет выглядеть как на Рис. 43.

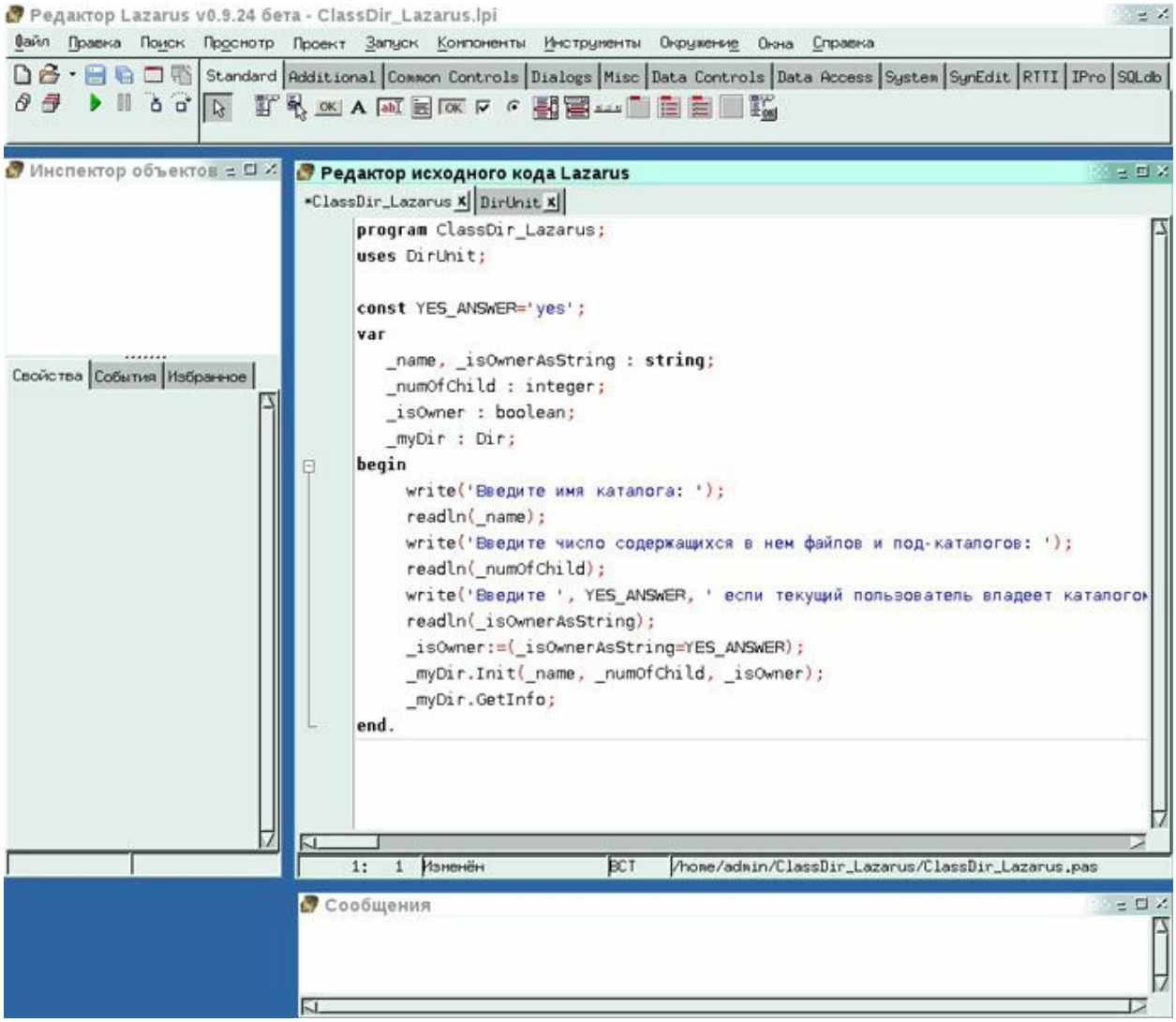

Рис. 43

30. Сохраните проделанную работу на диск для чего выберите пункты меню **Файл-Сохранить** (Рис. 41).

31. Попробуем скомпилировать нашу программу, для чего выберем в меню **Запуск-Собрать** или просто нажмем **Ctrl+F9** (Рис. 44).

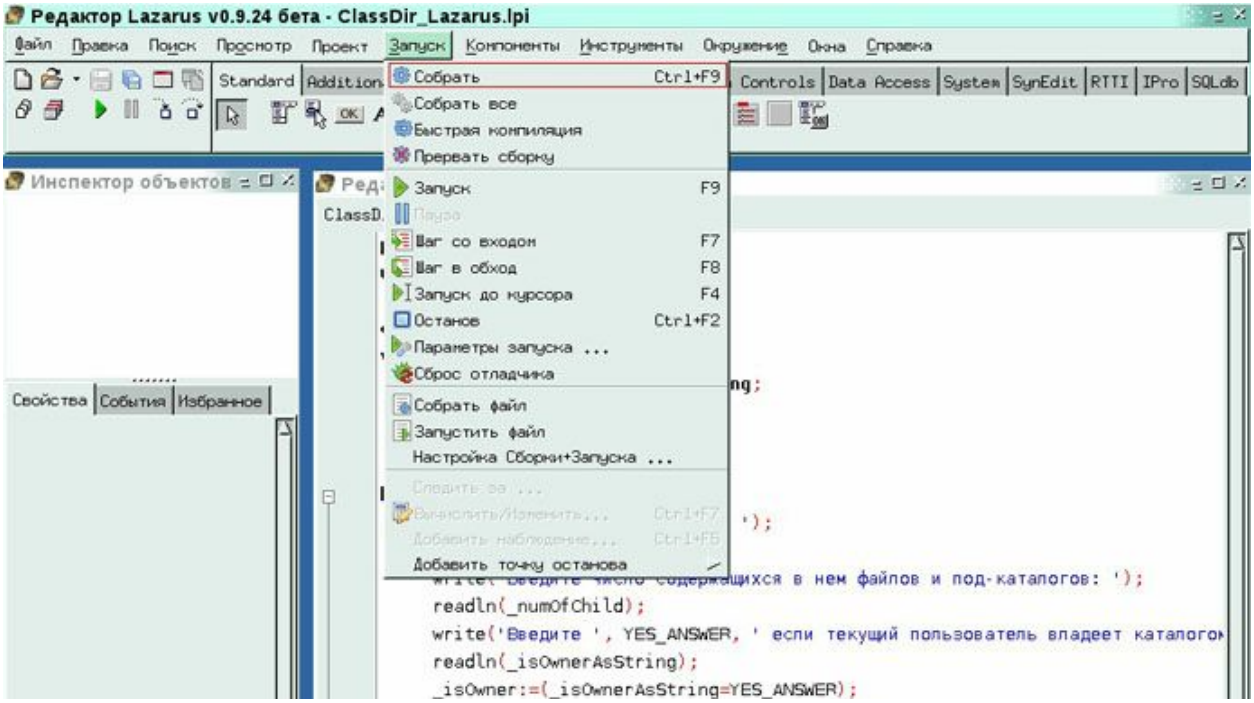

Рис. 44

32. Если никаких ошибок допущено не было окно **Сообщения** информирует нас об успешной сборке проекта (Рис. 45).

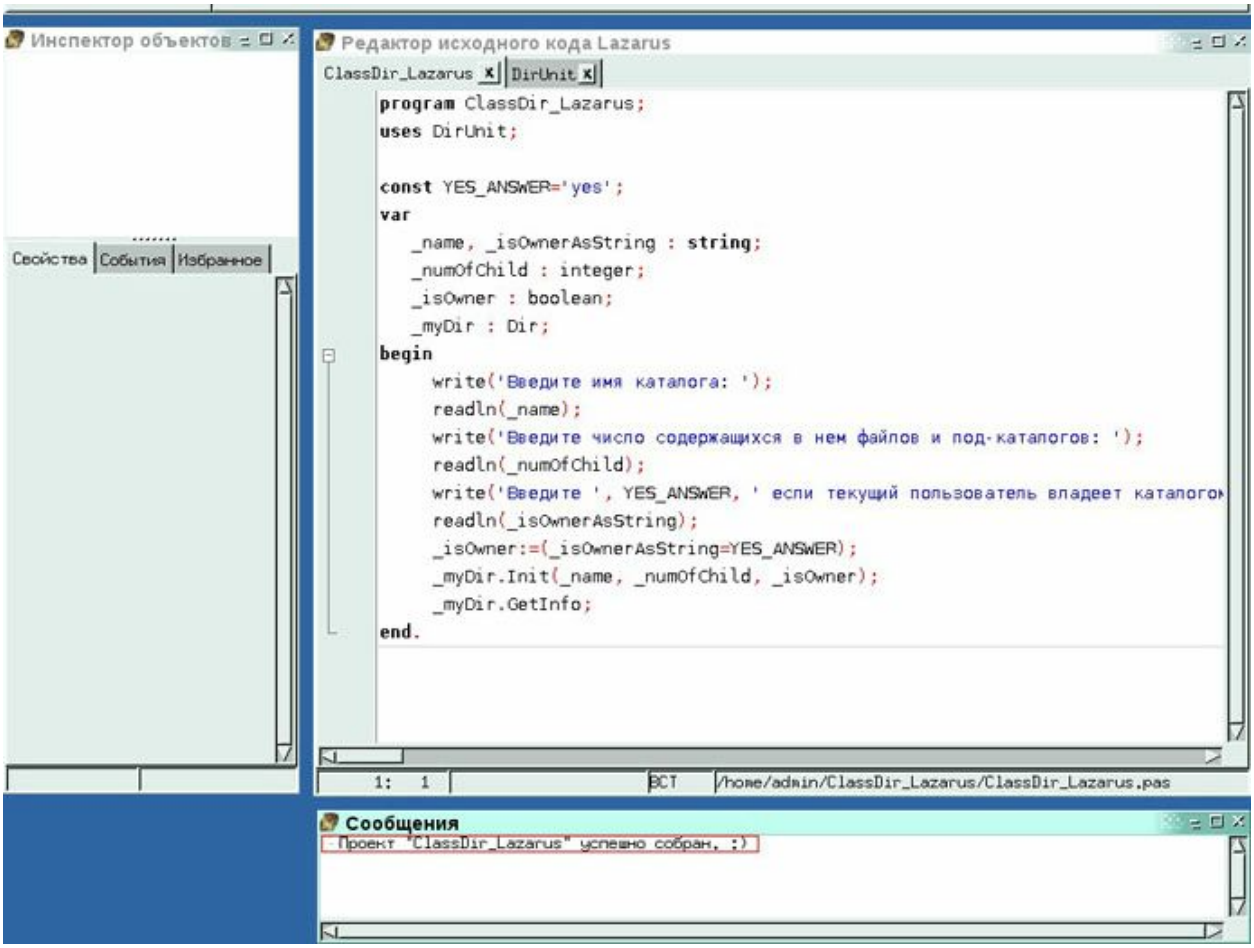

Рис. 45

33. Мы готовы произвести пробный запуск нашей программы, однако консоль, открываемую изнутри Lazarus нужно предварительно настроить. Выбираем в меню **Запуск-Параметры запуска…** (Рис. 46).

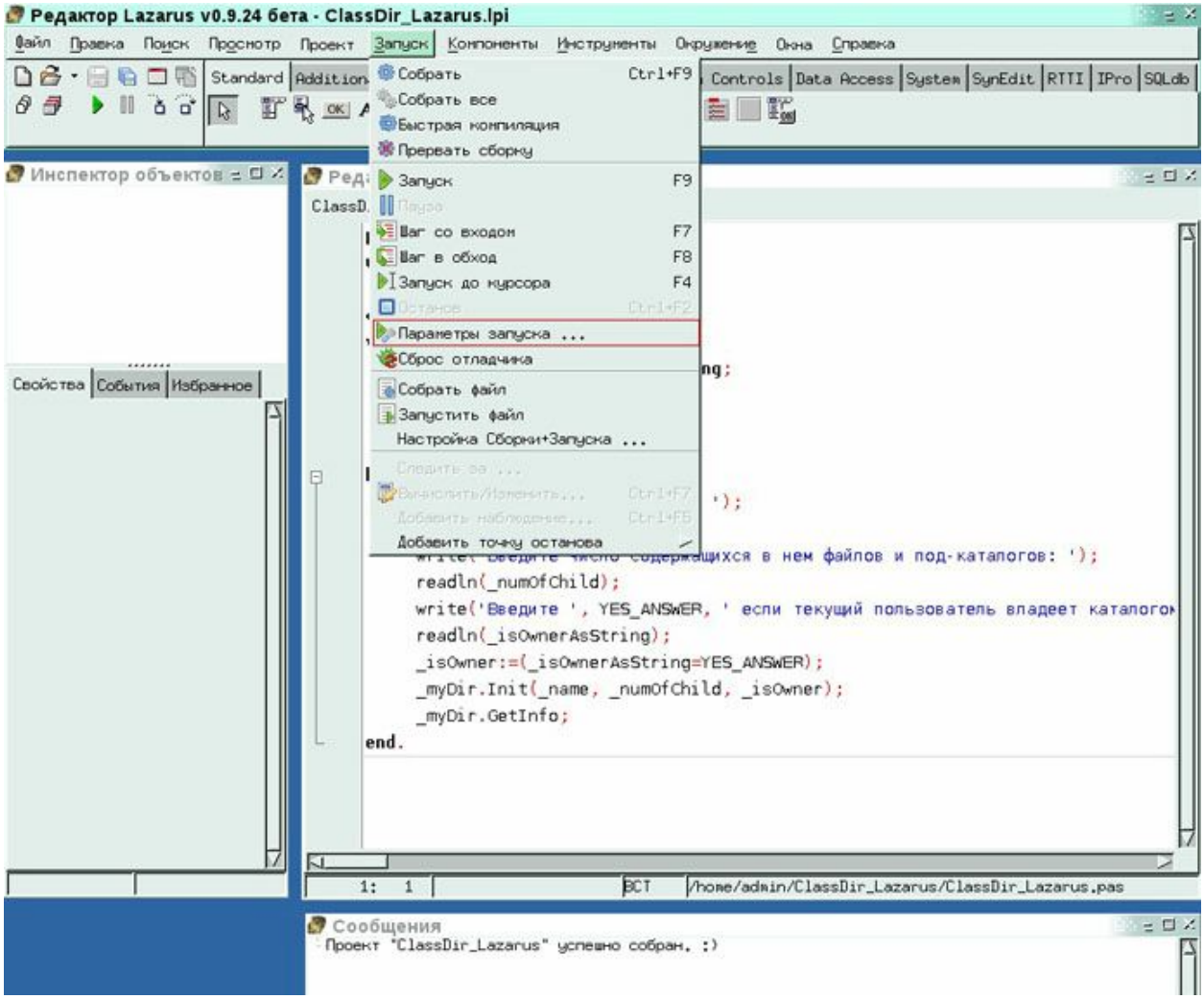

Рис. 46

34. В открывшемся диалоге **Параметры запуска** отмечаем опцию **Использовать приложение для запуска**. Помимо этого, в текстовом поле ниже этой опции надо проверить путь к приложению для запуска. По умолчанию таковым выступает **xterm** и среда предлагает такой путь к этому эмулятору терминала:

```
/usr/X11R6/bin/xterm -T 'Lazarus Run Output' -e $(LazarusDir)/tools/runwait.sh $(TargetCmdLine)
```

Здесь параметры **xterm** указаны верно, а вот путь к самому xterm ошибочен, т.к. он располагается по абсолютному пути **/usr/bin/xterm.** Поэтому исправляем вышеприведенную строку запуска убирая в ней ошибочную части пути (X11R6). Финальный вариант командной строки будет такой:

```
/usr/bin/xterm -T 'Lazarus Run Output' -e $(LazarusDir)/tools/runwait.sh $(TargetCmdLine)
```

Завершаем работу с данным диалогом нажатием на **OK** (Рис. 47).

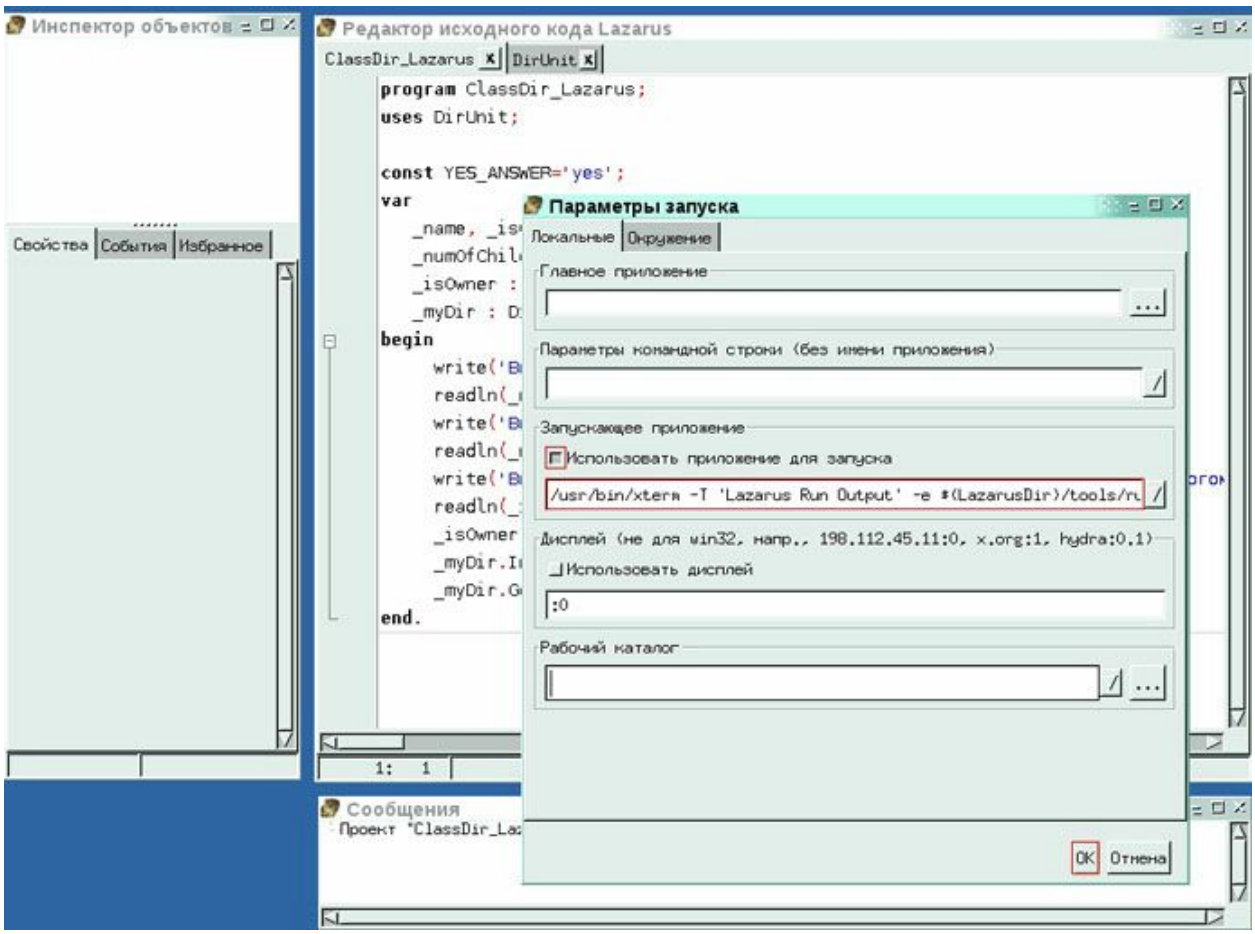

Рис. 47

35. Теперь мы готовы к тестовому прогону нашего приложения. Выберите в меню **Запуск-Запуск** или просто нажмите F9 (Рис. 48).

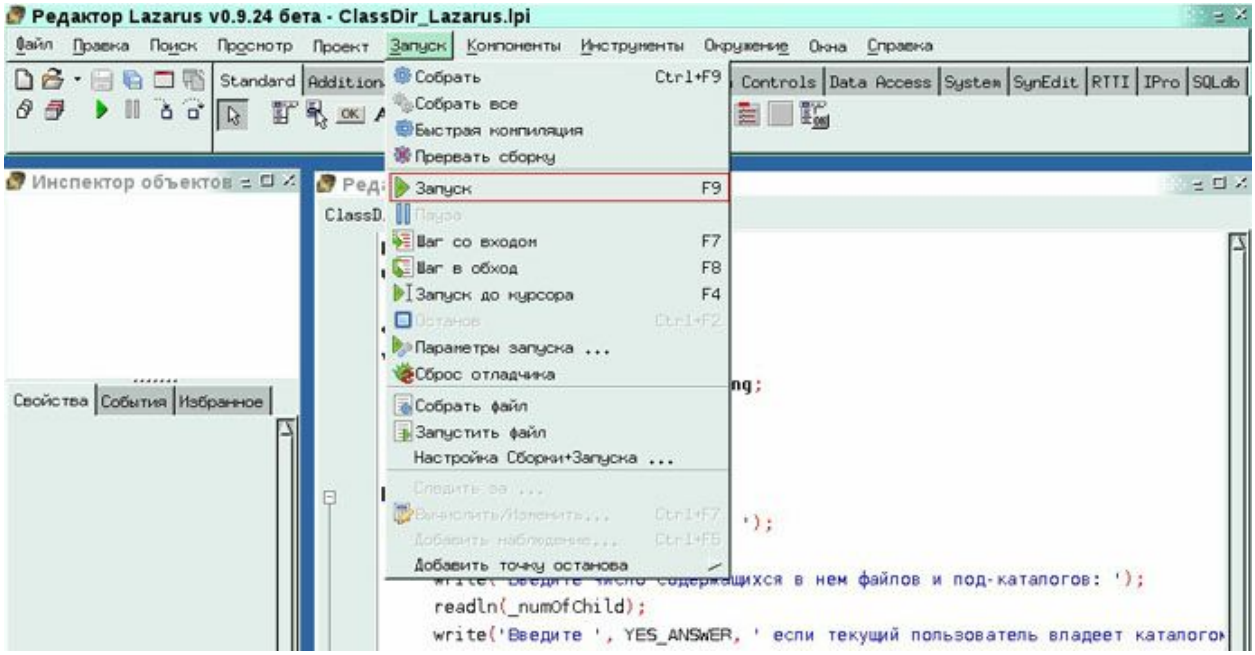

Рис. 48

36. Примите на себя роль пользователя программы **ClassDir_Lazarus** и ответьте на три вопроса подходящими сообщениями. Проанализируйте корректность вывода

информации на консоль методом GetInfo() класса **Dir**. Пример диалога с нашей программой представлен на Рис. 49. В этом диалоге в качестве ответов на вопросы были использованы строки «MyDocs», «187», «yes».

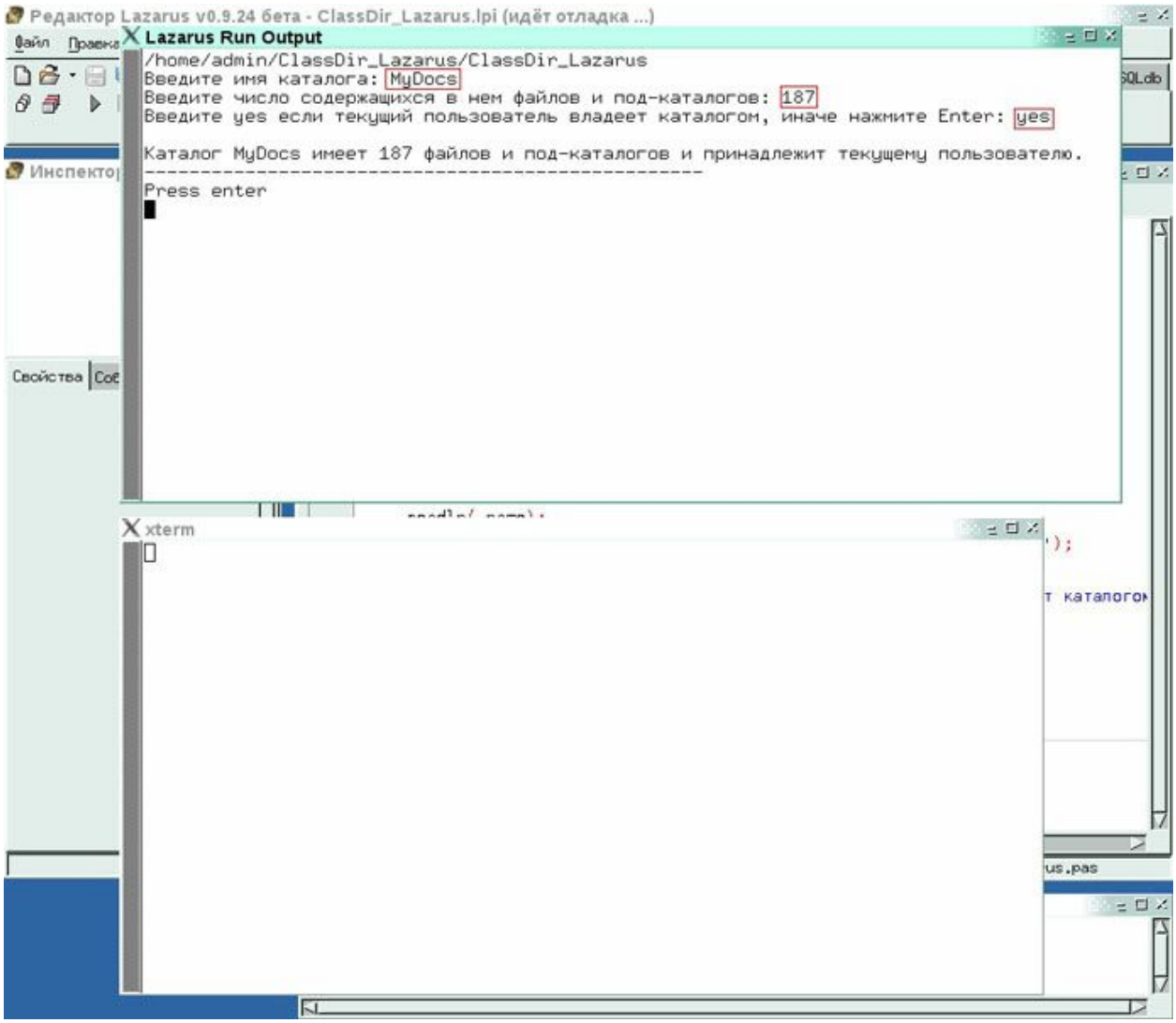

Рис. 49

Как предписывает нам последняя строчка в консольной сессии (Рис. 49) нажмите **Enter** что бы завершить ее и вернуться в среду разработки Lazarus. В информационном окне «Выполнение остановлено» нажмите единственную кнопку **OK** (Рис. 50).

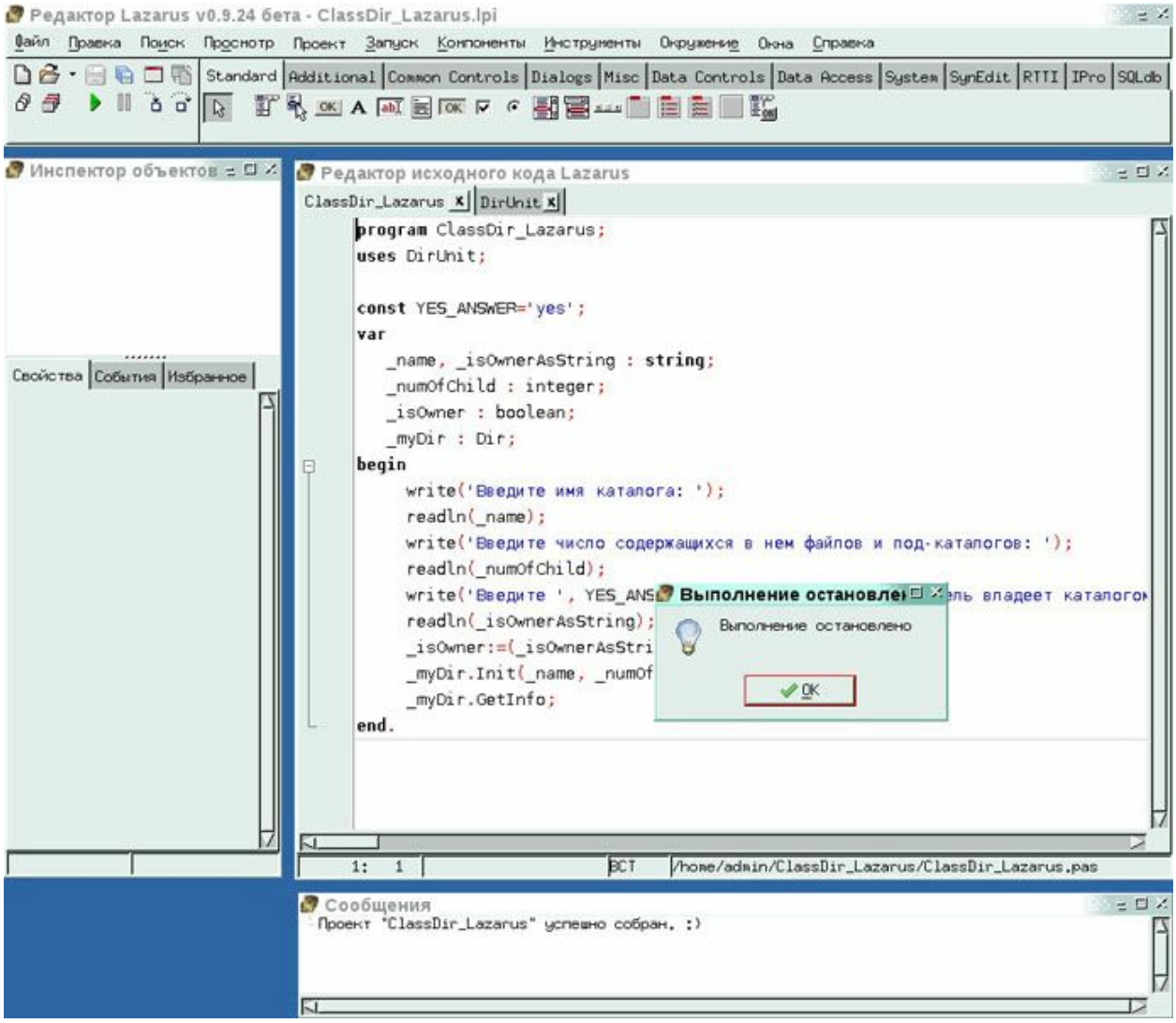

Рис. 50

37. Работа над проектом окончена. Закройте Lazarus с помощью меню **Файл-Выход** (Рис. 51)

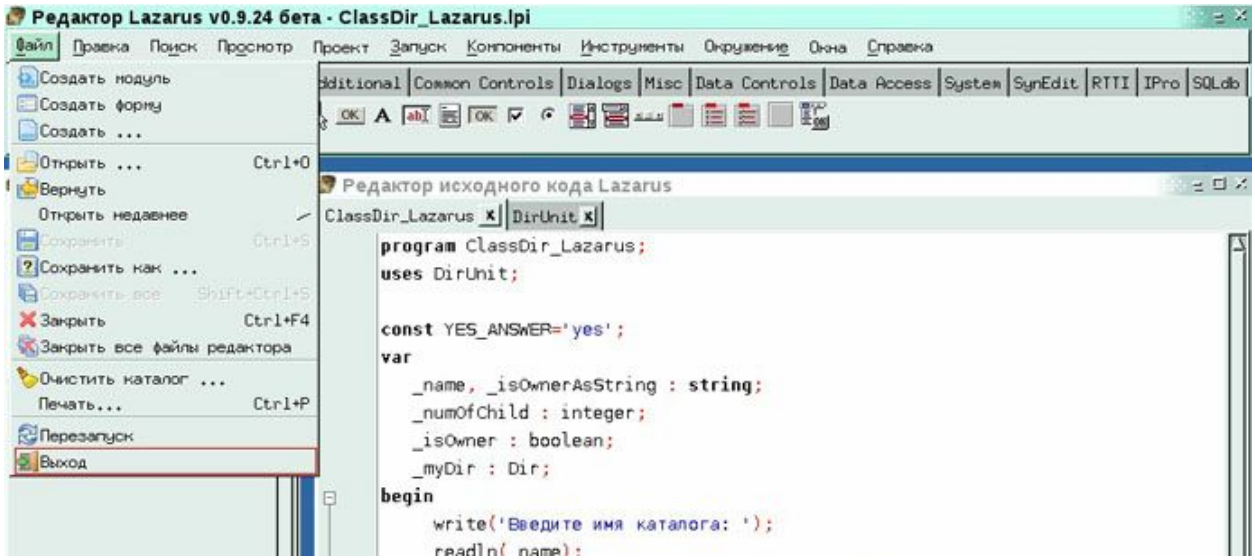

Рис. 51

38. Посмотрим где на жестком диске располагается скомпилированный нами исполнимый модуль. Это позволит нам распространить нашу программу для всех желающих воспользоваться ее функционалом. Откройте универсальный браузер Conqueror и перейдите в наш домашний каталог (/home/admin). В списке содержащихся в нем подпапок можно видеть и корневую папку нашего проекта - **Classier_Lazarus** (Рис. 52)

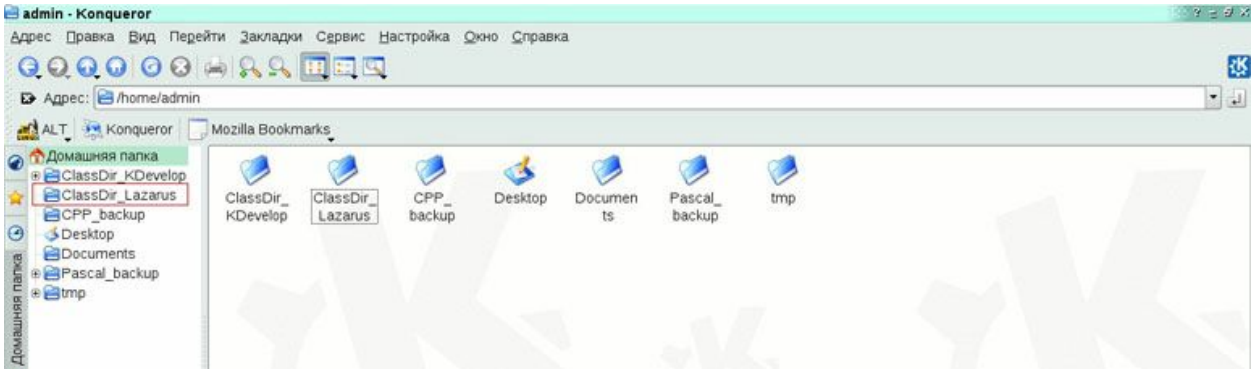

Рис. 52

39. В дереве папок слева выберите папку **ClassDir_Lazarus**. В ее содержимом справа можно видеть одноименную иконку приложения (Рис. 53). Это единственный файл, который мы можем/должны распространить заинтересованным лицам. Запуск его из командной строки консоли приведет к инициированию только что опробованному нами диалогу из трех вопросов и одного информационного сообщения. Так же заметьте, что в этой же папке собраны и все файлы исходного кода нашего приложения, а так же файл самого проекта (ClassDir_Lazarus.**lpi**).

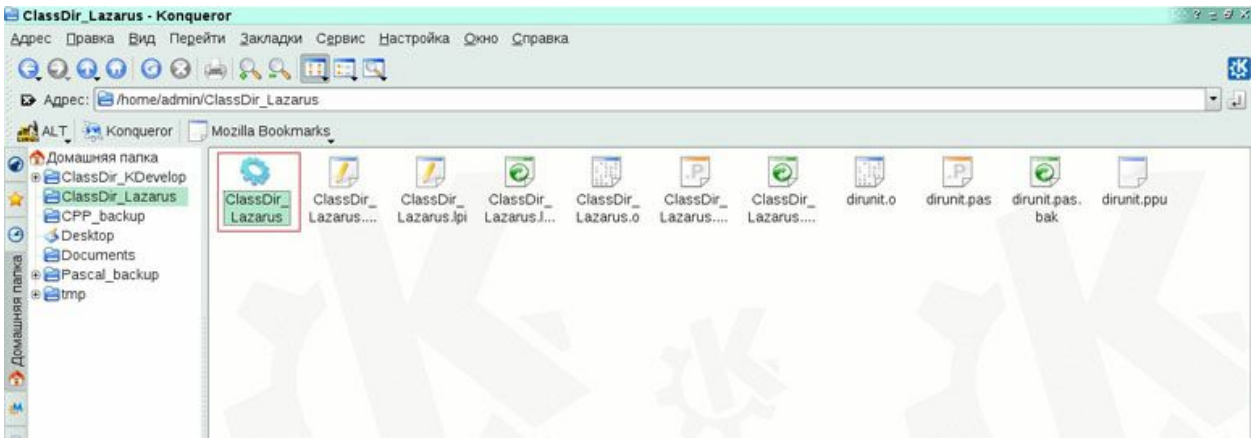

Рис. 53

3. Практика работы с Gambas и языком программирования BASIC

1. Снова целью этого практического занятия будет решение той же самой задачи обозначенной в разделе **«Постановка задачи»**, однако на этот раз в среде Gambas и языке BASIC.

Традиционно убедитесь, что вы вошли в систему с именем пользователя (login) **admin**. После этого из главного меню **КДЕ-Прочие-Разработка-Интегрированная среда разработки Gambas** (Gambas IDE) запустите среду разработки Gambas (Рис. 54).

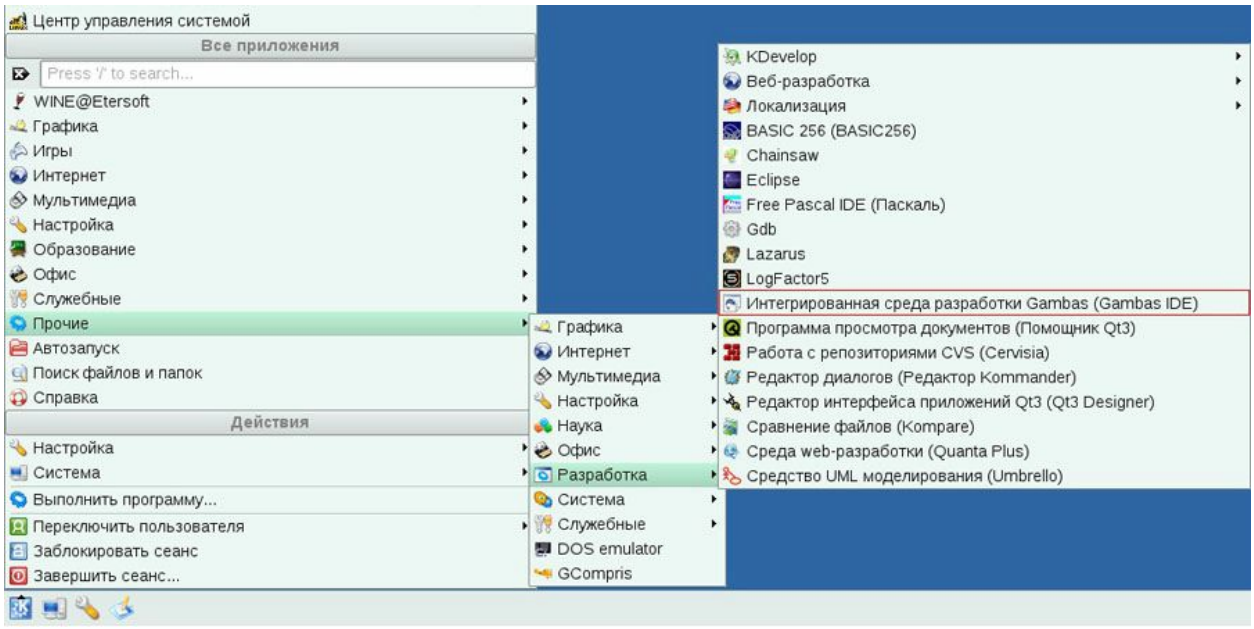

Рис. 54

2. В открывшемся окне **«Добро пожаловать в Gambas II»** выберите пункт **«Новый проект»** (Рис. 55).

Рис. 55

3. Это приведет к запуску трехшагового мастера создания нового проекта. На шаге **«1. Тип проекта»** укажите *Консольное приложение* и нажмите **Next** (Рис. 56).

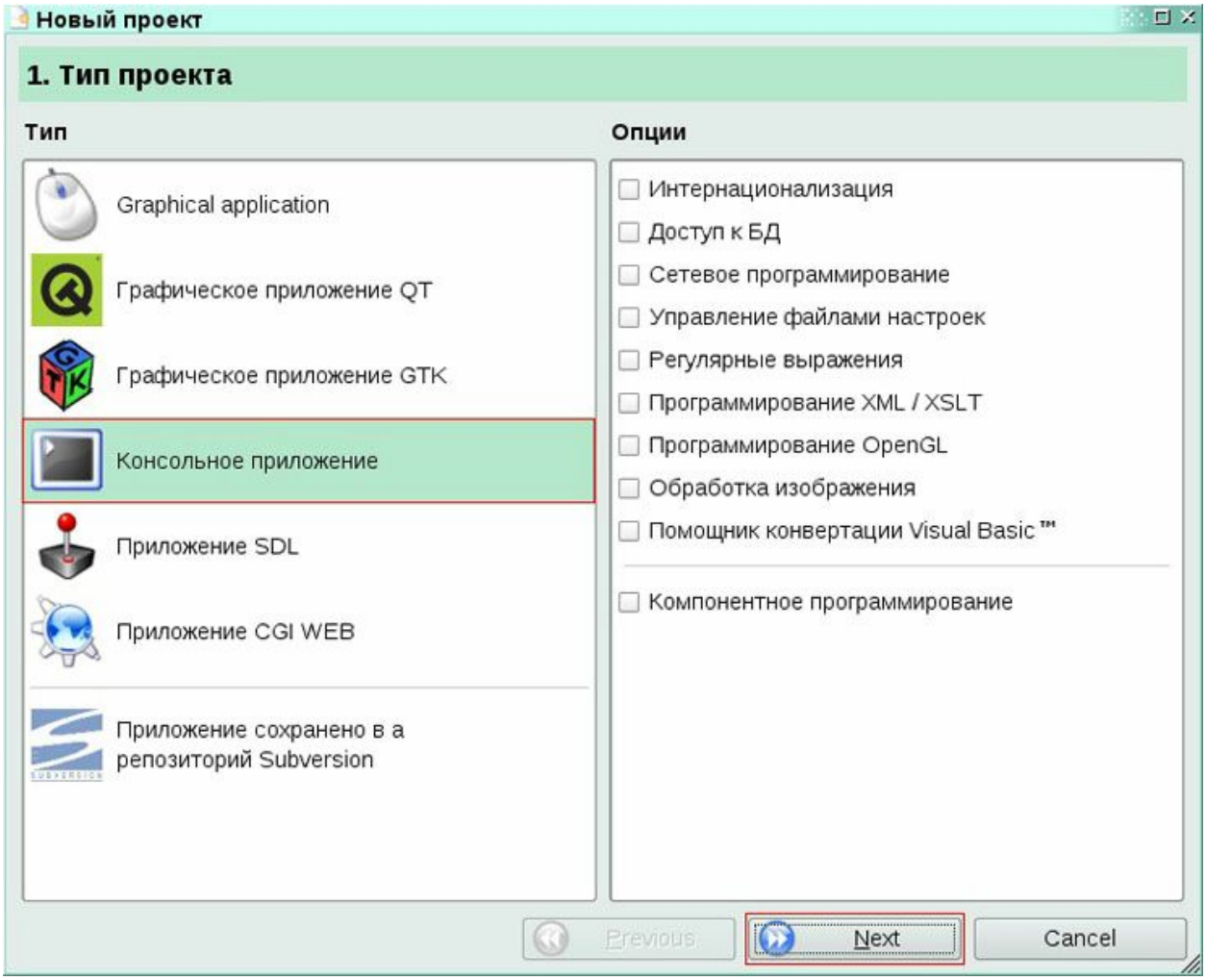

Рис. 56

4. На шаге **«2. Директория проекта»** мастера создания нового проекта мы можем выбрать родительскую папку для будущей корневой папки проекта. Нажатием на кнопку **Next** примите вариант по умолчанию (/home/admin) (Рис. 57).

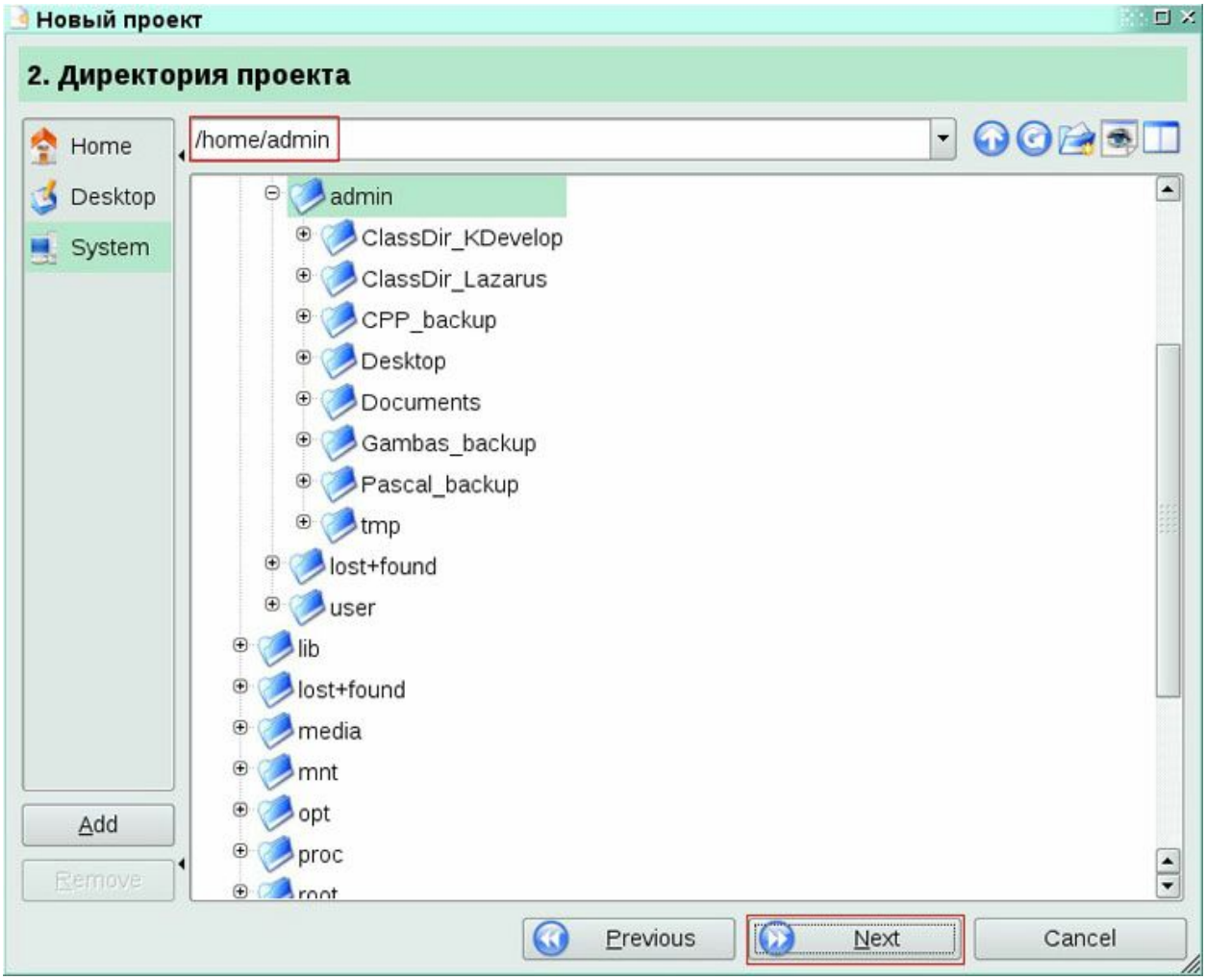

Рис. 57

5. На шаге финальном **«3. Информация о проекте»** мастера создания нового проекта укажите в поле **Имя** создаваемого проекта - **ClassDir_Gambas.** Напомним, что такое же имя будет иметь корневая папка проекта. Поле **Заголовок** оставьте пустым и нажмите **OK** (Рис. 58).

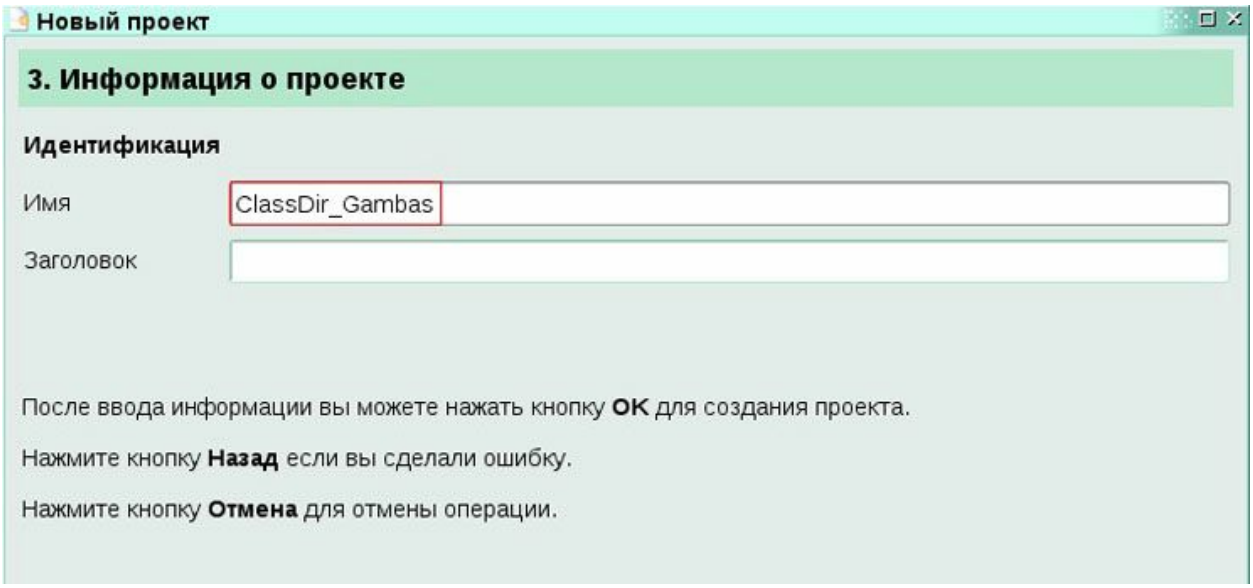

Рис. 58

6. В окне подтверждения создания нового проекта нажмите единственную кнопку **OK** (Рис. 59).

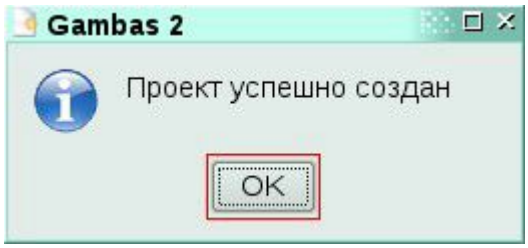

Рис. 59

7. Открывается рабочее пространство (workspace) пакета Gambas с только что созданным проектом. Проект пока состоит из единственного файла - **MMain.module**. Это файл, содержащий код главной подпрограммы Main. Однако среда не открывает его для редактирования автоматически. Мы сделаем это самостоятельно чуть позже. Пока в окне **«Совет дня»** очистите флажок **Показывать советы при старте** и нажмите **Закрыть** (Рис. 60).

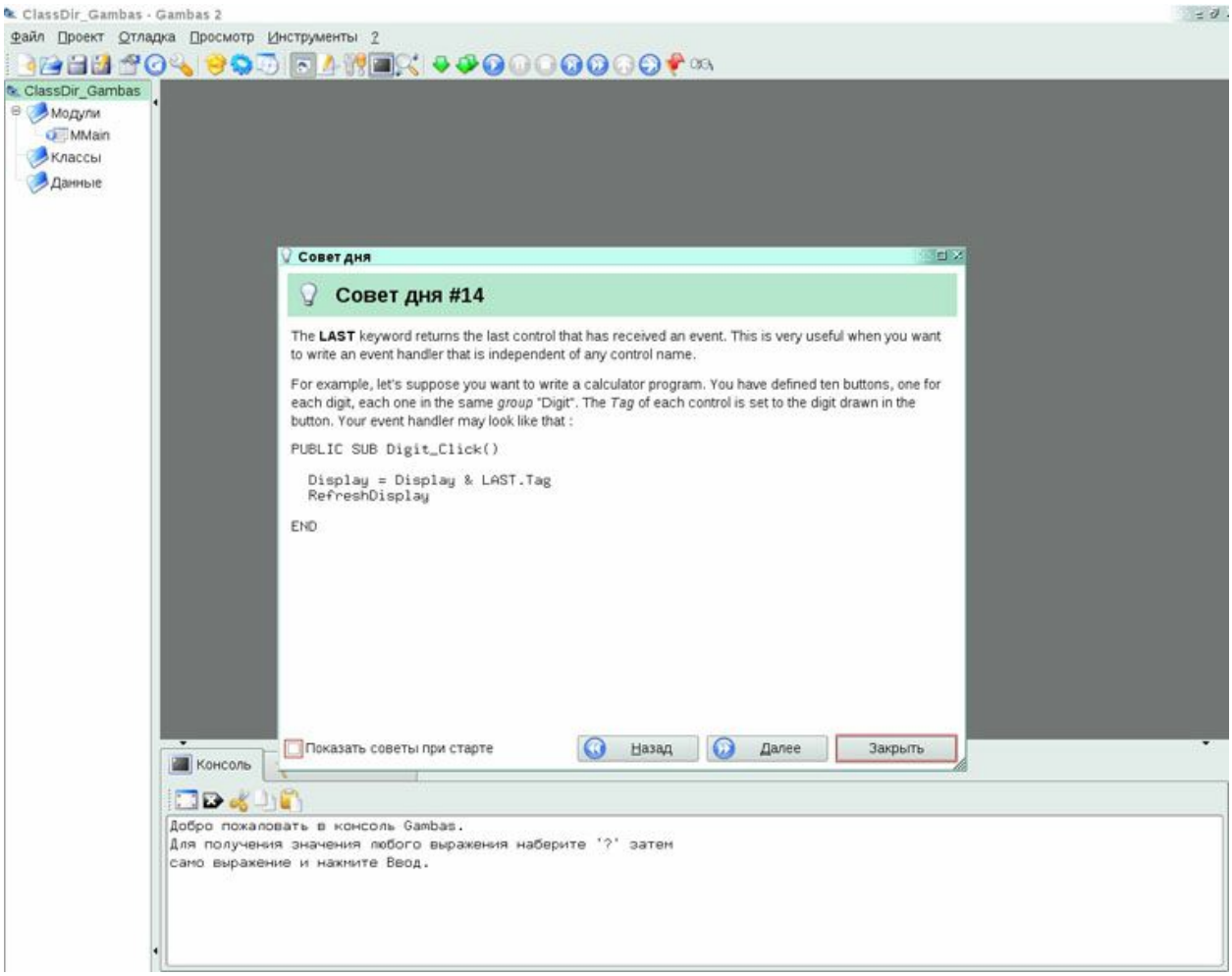

Рис. 60

8. Давайте приступим к написанию кода. Однако, как и в двух предыдущих практиках, начнем не с файла главной подпрограммы, а с несуществующего пока файла

класса **Dir**. Добавим его в наш проект, для чего щелкнем правой кнопкой мыши в любом месте вспомогательного окна проекта и выберем из появившегося контекстного меню **Новый-Класс…** (Рис. 61).

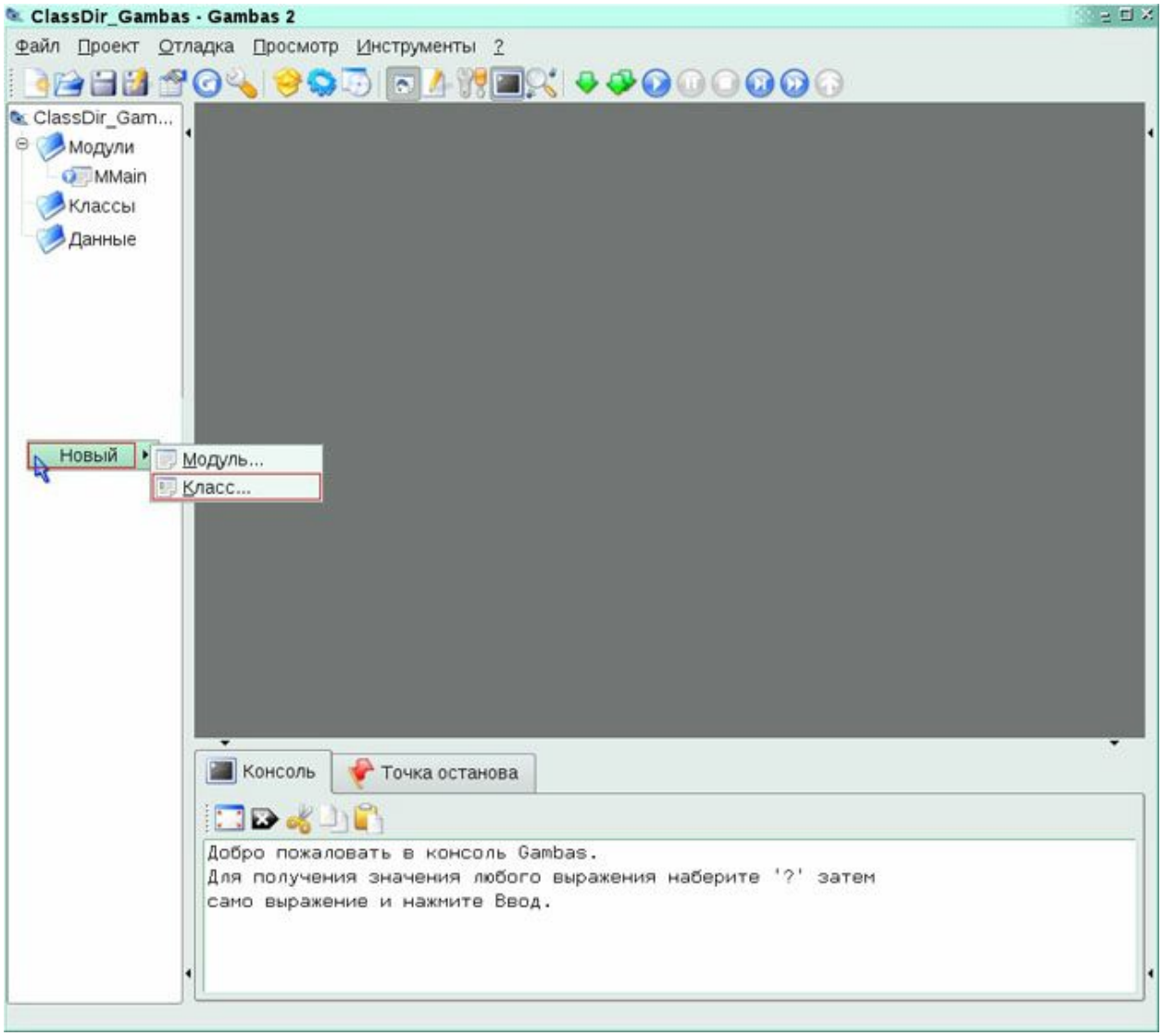

Рис. 61

9. Появляется окно диалога **«Новый файл»**. В нем в поле Имя укажите имя нашего будущего класса - **Dir**, и нажмите **OK** (Рис. 62).

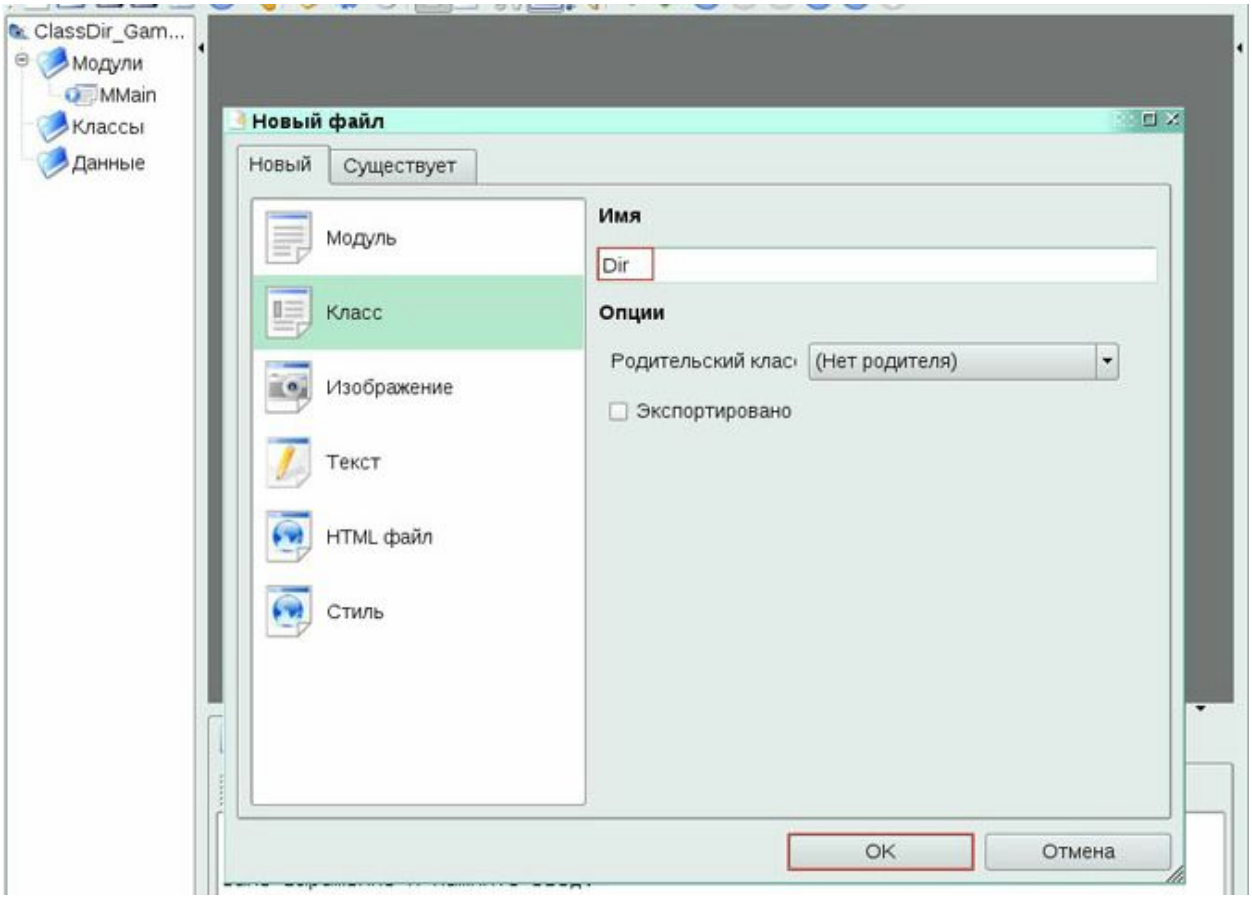

Рис. 62

10. Новый файл добавляется к проекту и визуально отображается в окне проекта под узлом дерева **Классы**. (Рис. 63) Так же новый файл автоматически открывается для редактирования. Пока он содержит всего лишь одну строку комментария:

```
' Gambas class file
```

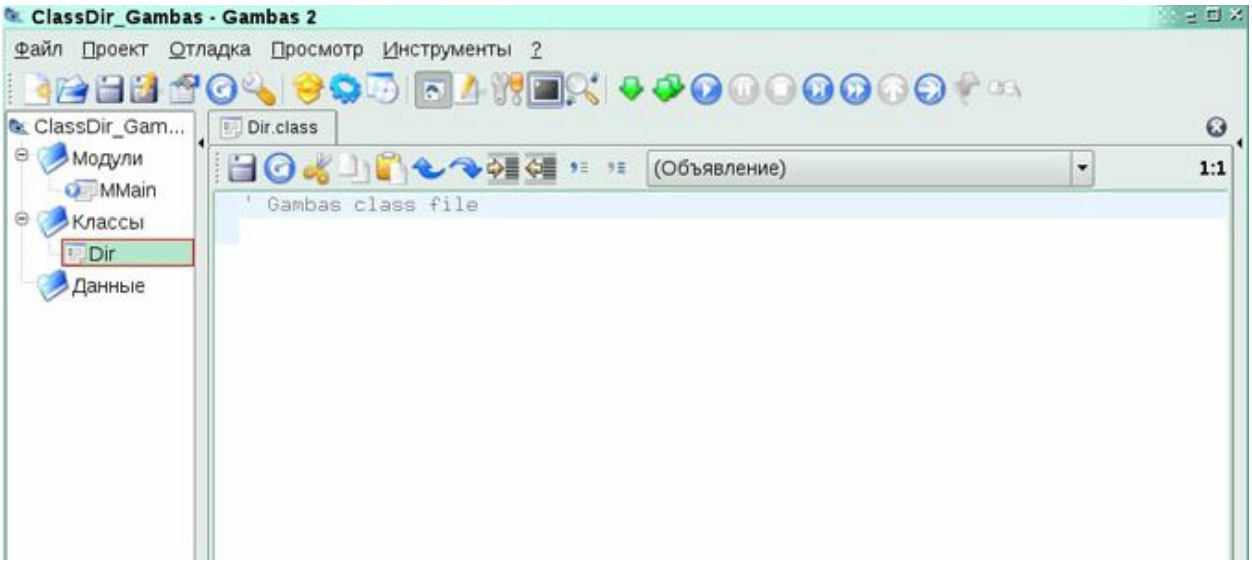

Рис. 63

11. Оставим комментарий как он есть. Начиная со строки 2, задекларируйте все 3 поля нашего класса **Dir.** Соблюдая принципы ООП, объявите их приватными:

```
PRIVATE _name AS String
PRIVATE _numOfChild AS Integer
PRIVATE _isOwner AS Boolean
```

12. С целью структуризации текста оставьте одну строчку пустой, а затем декларируйте и реализуйте конструктор нового класса:

```
PUBLIC SUB _new(name AS String, numOfChild AS Integer, isOwner AS Boolean)
  _name = name
  _numOfChild = numOfChild
  _isOwner = isOwner
END
```

13. Точно так же поступите со вторым методом (подпрограммой) класса **Dir**, GetInfo:

```
PUBLIC SUB GetInfo()
  PRINT
  PRINT "Каталог " & _name & " имеет " & _numOfChild & " файлов и под-каталогов и ";
  IF _isOwner THEN
    PRINT "принадлежит текущему пользователю."
  ELSE
    PRINT "НЕ принадлежит текущему пользователю."
  ENDIF
END
```

14. Итоговый код файла Dir.class будет примерно таким:

```
' Gambas class file
PRIVATE _name AS String
PRIVATE _numOfChild AS Integer
PRIVATE _isOwner AS Boolean

PUBLIC SUB _new(name AS String, numOfChild AS Integer, isOwner AS Boolean)
  _name = name
  _numOfChild = numOfChild
  _isOwner = isOwner
END
PUBLIC SUB GetInfo()
  PRINT
  PRINT "Каталог " & _name & " имеет " & _numOfChild & " файлов и под-каталогов и ";
  IF _isOwner THEN
    PRINT "принадлежит текущему пользователю."
  ELSE
    PRINT "НЕ принадлежит текущему пользователю."
  ENDIF
END
```

А визуально, в редакторе, он будет выглядеть как на Рис. 64:

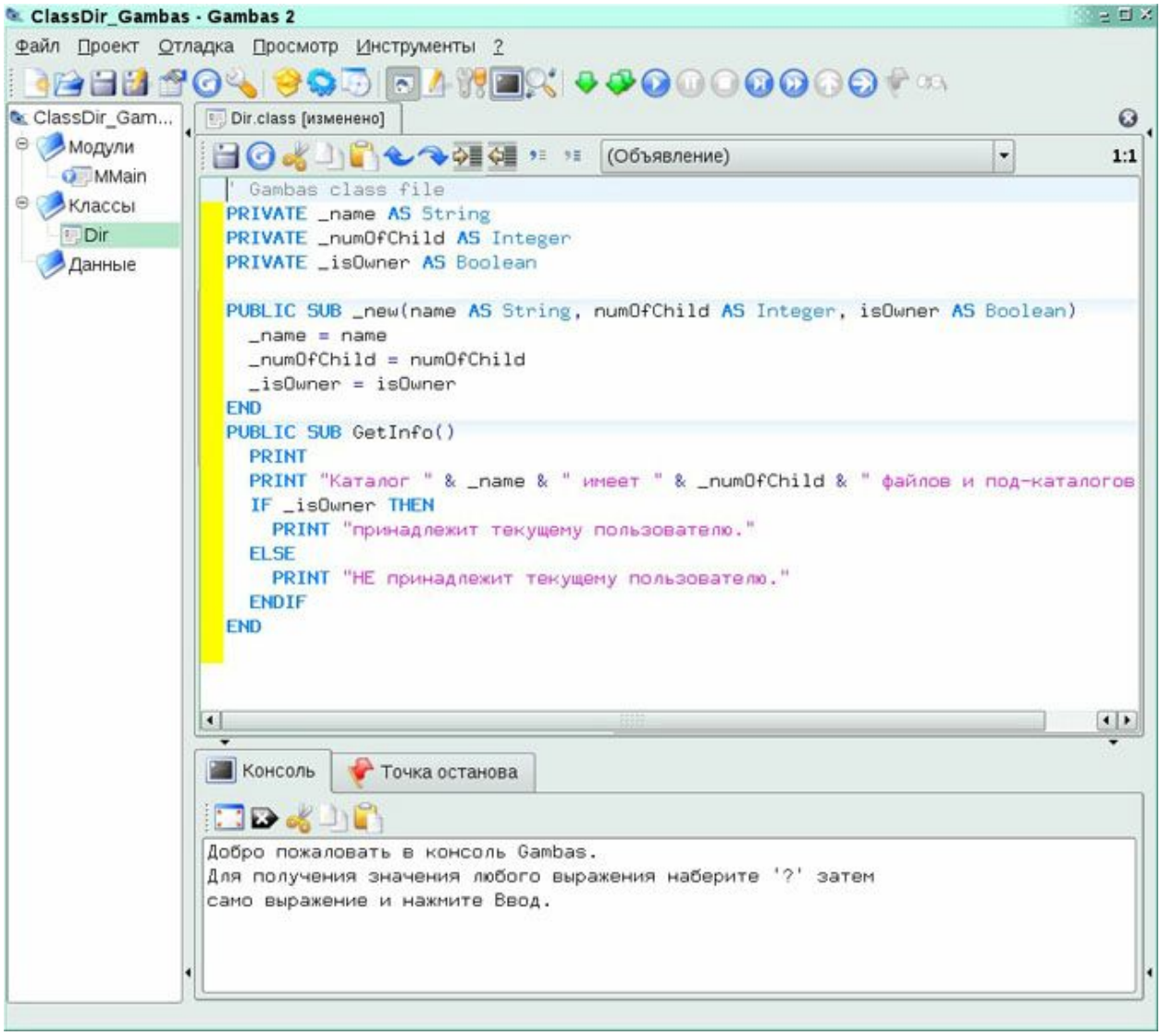

Рис. 64

15. Теперь двойным щелчком по имени файла главной подпрограммы - **MMain** - в окне проекта откроем этот файл для редактирования. Пока код этого файла содержит лишь комментарий и декларацию главной подпрограммы **Main**. «Запомним» этот стартовый код для возможности вернуться к нему:

```
' Gambas module file

PUBLIC SUB Main()

END
```

Визуально же все пока выглядит как на Рис. 65.

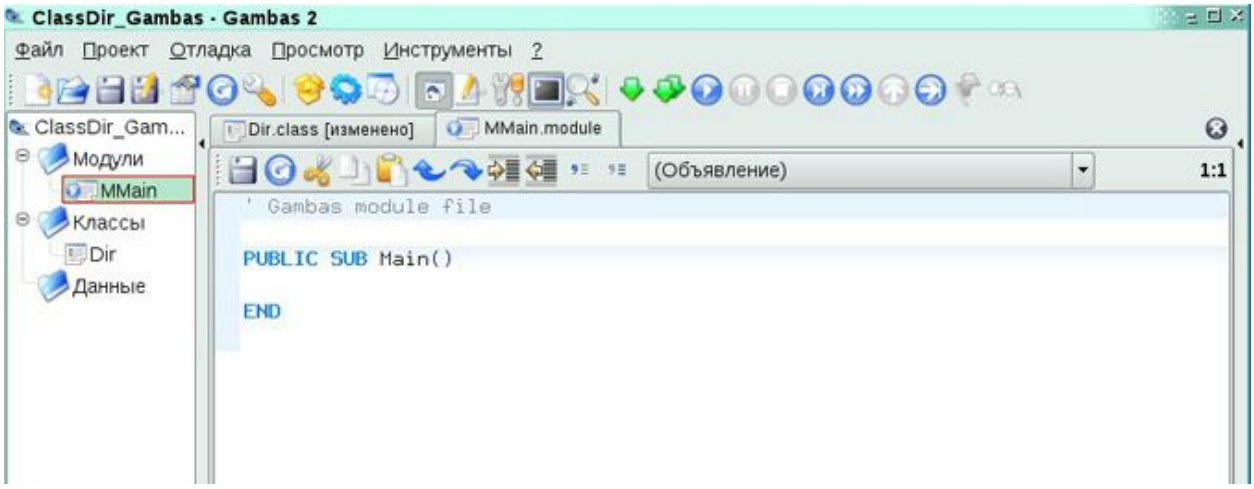

Рис. 65

16. В строке 2, между комментарием и декларацией подпрограммы Main, объявите требуемую по условиям разрабатываемой программы константу:

```
CONST YES_ANSWER AS String = "yes"
```

17. На строчке следующей за декларацией подпрограммы Main объявите все пять локальных переменных требуемых этой подпрограмме для нормальной работы:

```
DIM _name, _isOwnerAsString AS String
DIM _numOfChild AS Integer
DIM _isOwner AS Boolean
DIM _myDir AS Dir
```

18. Далее запросите у пользователя нашей программы всю необходимую информацию:

```
PRINT "Введите имя каталога: "
INPUT _name
PRINT "Введите число содержащихся в нем файлов и подкаталогов: "
INPUT _numOfChild
PRINT "Введите " & YES_ANSWER & " если текущий пользователь владеет каталогом, иначе нажмите Enter: "
INPUT _isOwnerAsString
```

19. Проанализируйте пользовательский ввод для последней переменной и на этом основании установите другую переменную, **_isOwner**, в значение *true* либо *false*:

```
_isOwner = (_isOwnerAsString = YES_ANSWER)
```

20. Наконец создайте экземпляр класса **Dir** вызовом его конструктора с требуемым числом аргументов. На этом новом экземпляре вызовите информационный метод (подпрограмму) **GetInfo:**

```
_myDir = NEW Dir(_name, _numOfChild, _isOwner)
_myDir.GetInfo
```

21. Итоговый код файла главной подпрограммы будет таким:

```
' Gambas module file
CONST YES_ANSWER AS String = "yes"
PUBLIC SUB Main()
DIM _name, _isOwnerAsString AS String
DIM _numOfChild AS Integer
```

```
        DIM _isOwner AS Boolean
        DIM _myDir AS Dir
             PRINT "Введите имя каталога: "
             INPUT _name
             PRINT "Введите число содержащихся в нем файлов и подкаталогов: "
             INPUT _numOfChild
             PRINT "Введите " & YES_ANSWER & " если текущий пользователь владеет
каталогом, иначе нажмите Enter: "
             INPUT _isOwnerAsString
             _isOwner = (_isOwnerAsString = YES_ANSWER)
             _myDir = NEW Dir(_name, _numOfChild, _isOwner)
             _myDir.GetInfo
        END
```

Визуально же итоговый результат должен выглядеть как на Рис. 66.

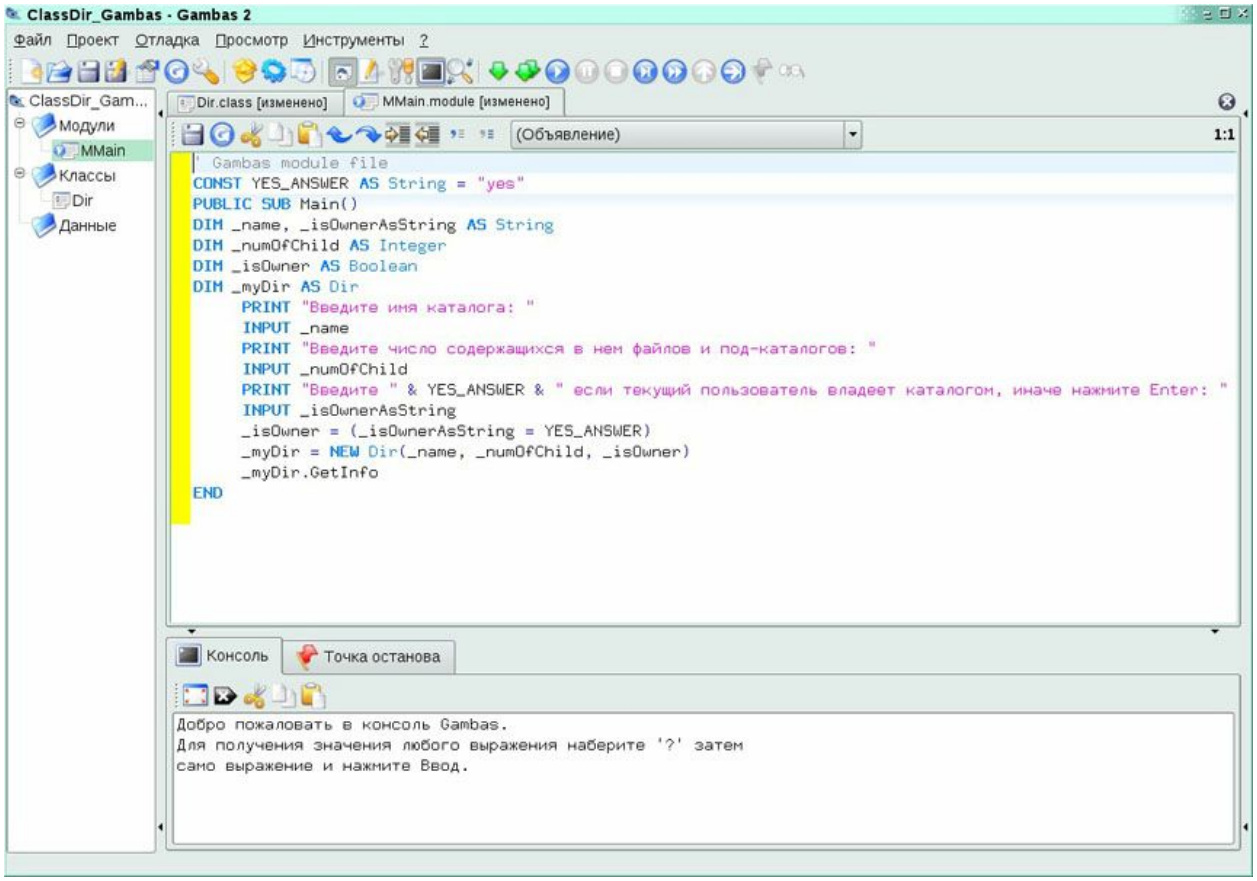

Рис. 66

22. Код программы написан. В целом было бы неплохой идеей сохранить результаты работы на диск. Однако мы знаем, что запуск процедуры компиляции сначала сохраняет все несохраненные файлы и лишь потом переходит непосредственно к компиляции. Воспользуемся этой удобной функциональностью «два в одном» и перейдем к указанной процедуре. Для этого достаточно выбрать в главном меню среды *Проект*-**Компилировать Все** или просто нажать **Alt+F7** (Рис. 67).

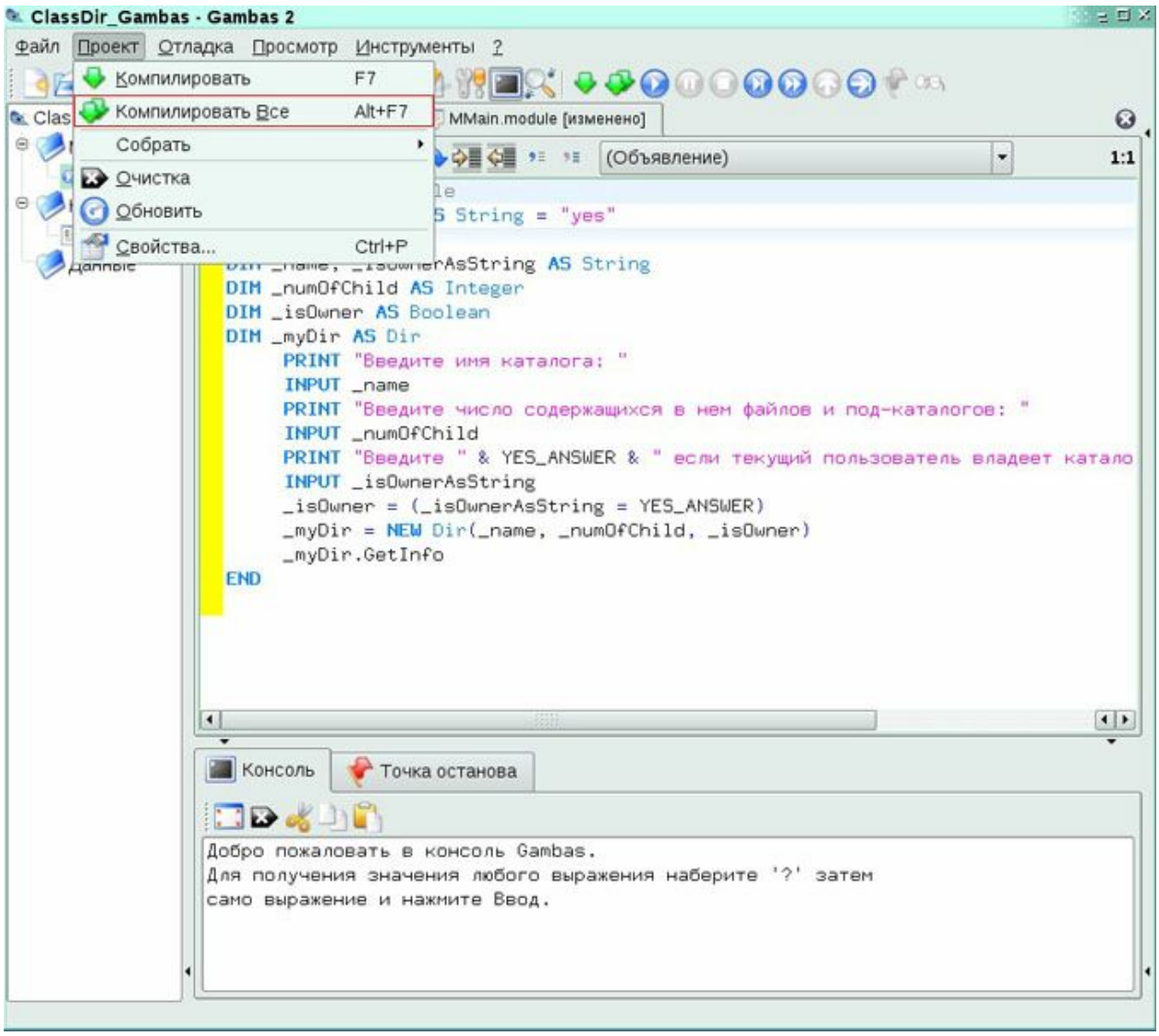

Рис. 67

23. Признаком успешной компиляции является отсутствие каких-либо информационных окон с сообщениями об ошибках. Если же такое окно возникло - прочтите сообщение об ошибке, исправьте ее в редакторе и перезапустите компиляцию проекта. Добейтесь успеха в этой операции, прежде чем двигаться дальше.

24. После успешной компиляции интерпретатор языка Gambas готов выполнить наш код строчка за строчкой. Напомним, что в отличии от большинства компиляторов и сред разработок компиляция программы на языке Gambas НЕ означает создания выходного исполнимого модуля. Однако для запуска в среде Gambas наша программа полностью готова, выбираем в главном меню среды **Отладка-Старт** или просто нажимаем **F5** (Рис. 68).

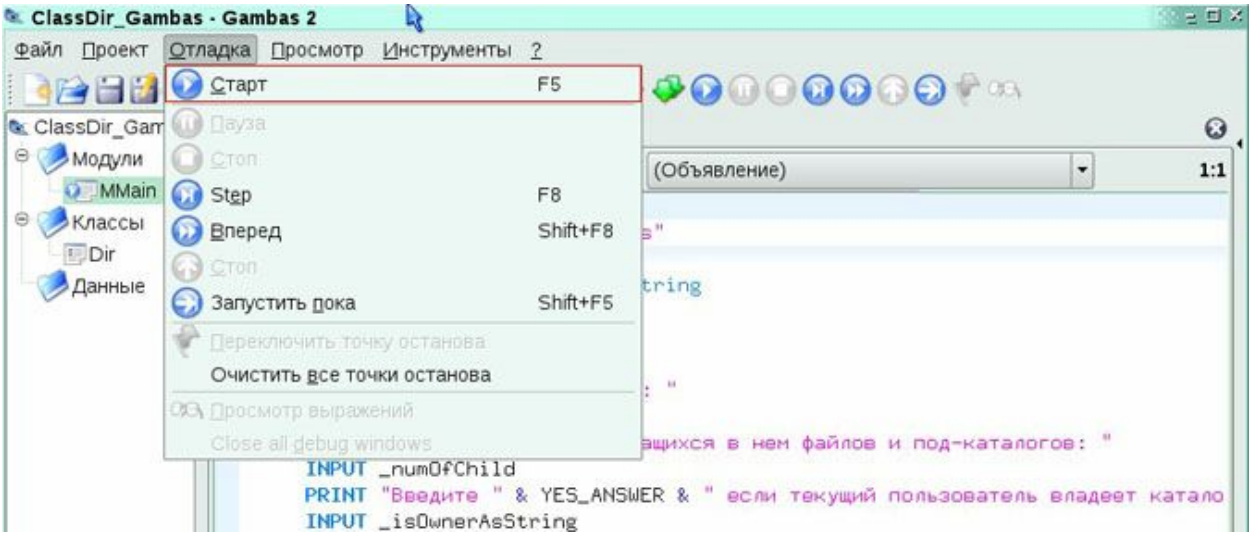

Рис. 68

25. Примите на себя роль пользователя программы **ClassDir_Gambas** и ответьте на три вопроса подходящими сообщениями. Проанализируйте корректность вывода информации на консоль методом GetInfo() класса **Dir**. Пример диалога с нашей программой представлен на Рис. 69. В этом диалоге в качестве ответов на вопросы были использованы строки «MyDocs», «187», «yes». Имейте в виду, что в среде Gambas окно консоли встроено прямо в главное окно и не требует открытия окна консоли реальной, как это было в двух предыдущих средах.

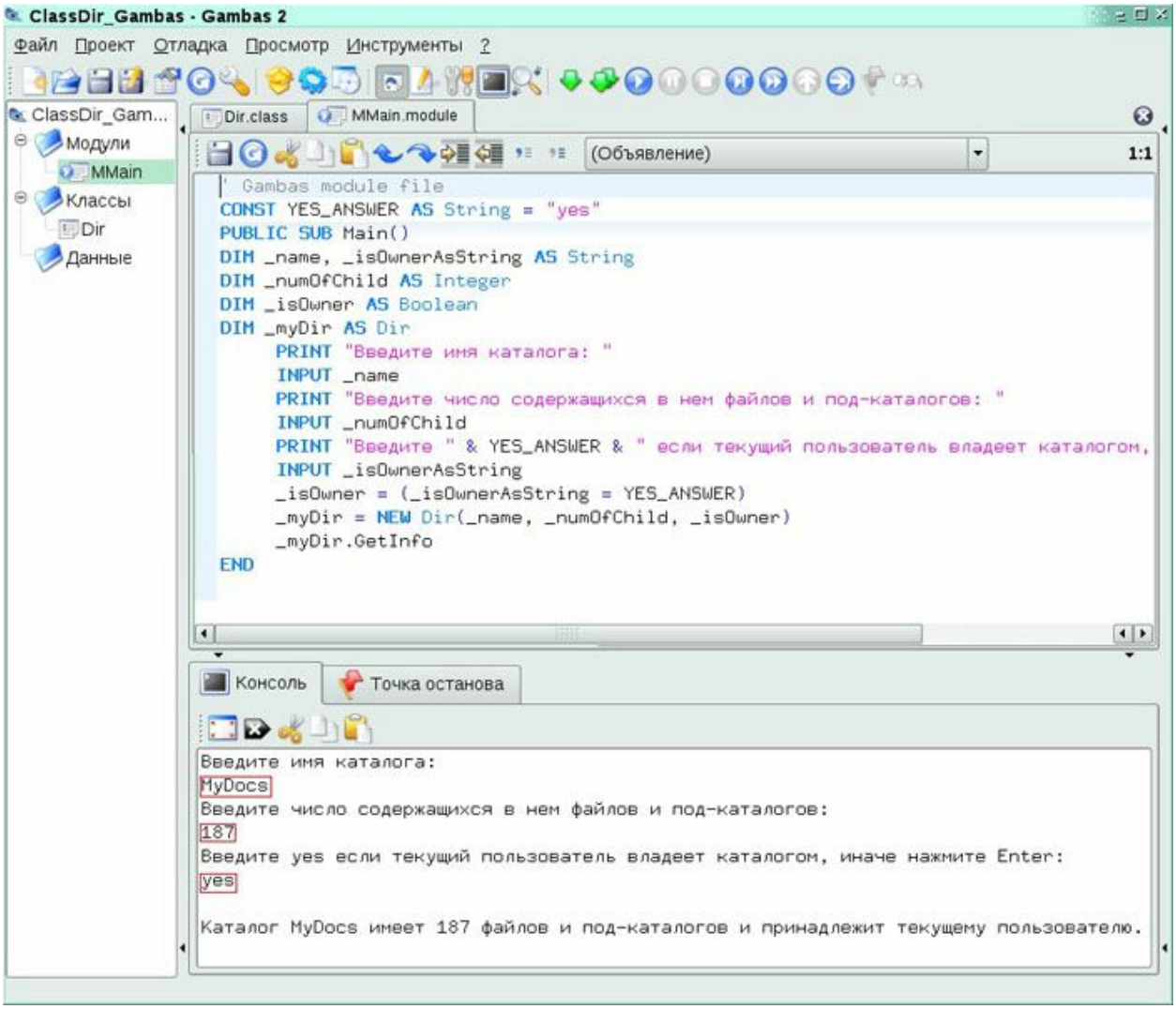

Рис. 69

26. Если нам требуется теперь передать готовую программу на другой компьютер то можно, безусловно, просто отдать файлы исходного кода. При наличии на целевом компьютере интерпретатора языка Gambas эти файлы можно будет запустить и там. Посмотрим более удобный способ распространения готовой программы - упаковка всех файлов исходного кода в один архив. Выберите в главном меню **Проект-Собрать-Executable…** или, что тоже самое, нажмите **Ctrl+Alt+M** (Рис. 70).

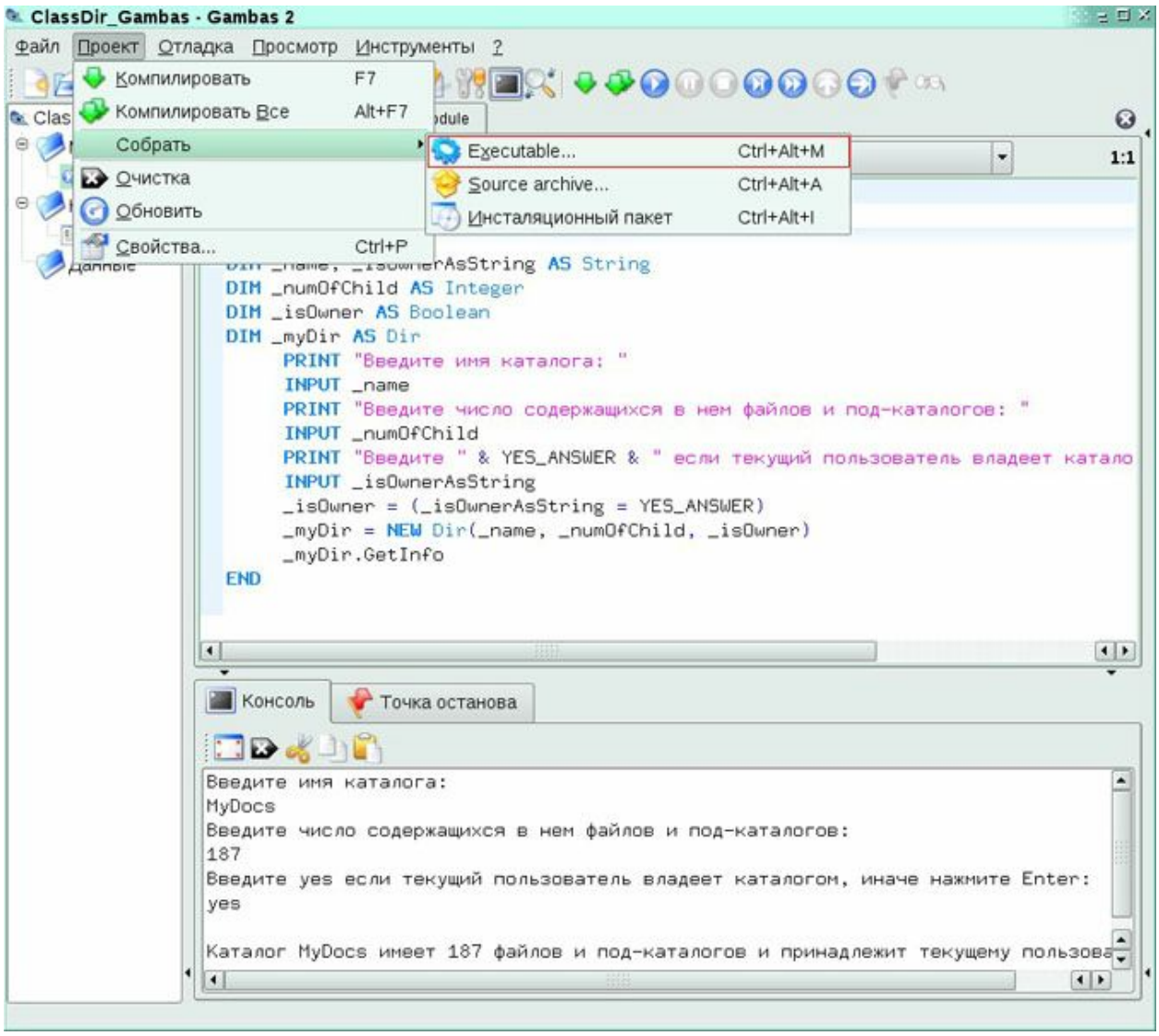

Рис. 70

27. В появившемся окне **«Сделать исполняемый модуль»** (название окна не слишком удачное; создаваемый модуль не будет таковым с т.з. операционной системы, а будет просто архивом файлов исходного кода) проверьте значения по умолчанию:

- папка для создания модуля - **/home/admin/ClassDir_Gambas**
- имя создаваемого модуля - **ClassDir_Gambas.gambas**

и нажмите **OK** (Рис. 71).

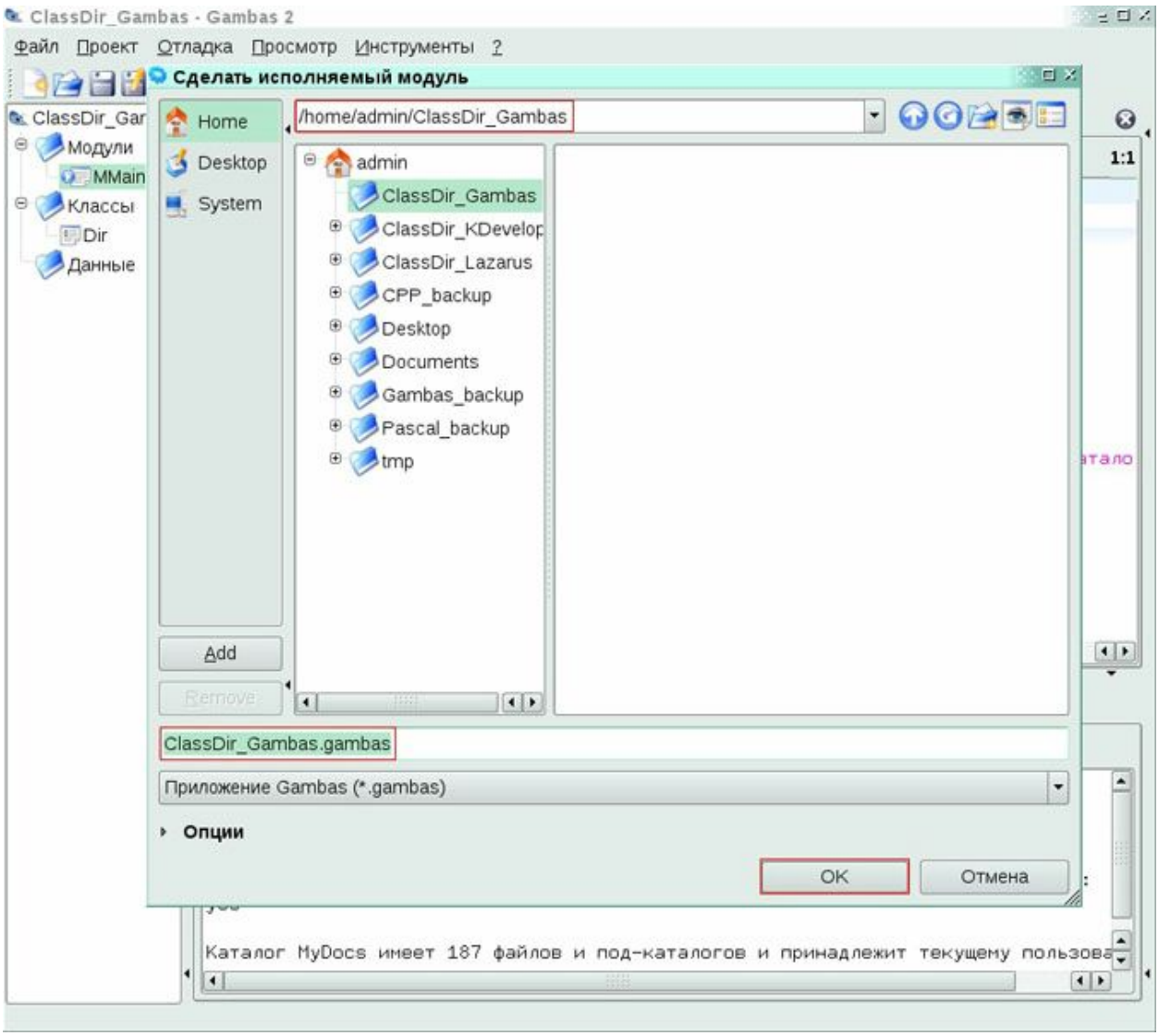

Рис. 71

28. Снова критерием успешности процесса будет отсутствие каких-либо информационных окон с сообщениями об ошибках в создании выходного пакета. Мы закончили работу со средой Gambas и можем ее закрыть. Как и везде - меню **Файл-Выход** или просто **Ctrl+Q** (Рис. 72).

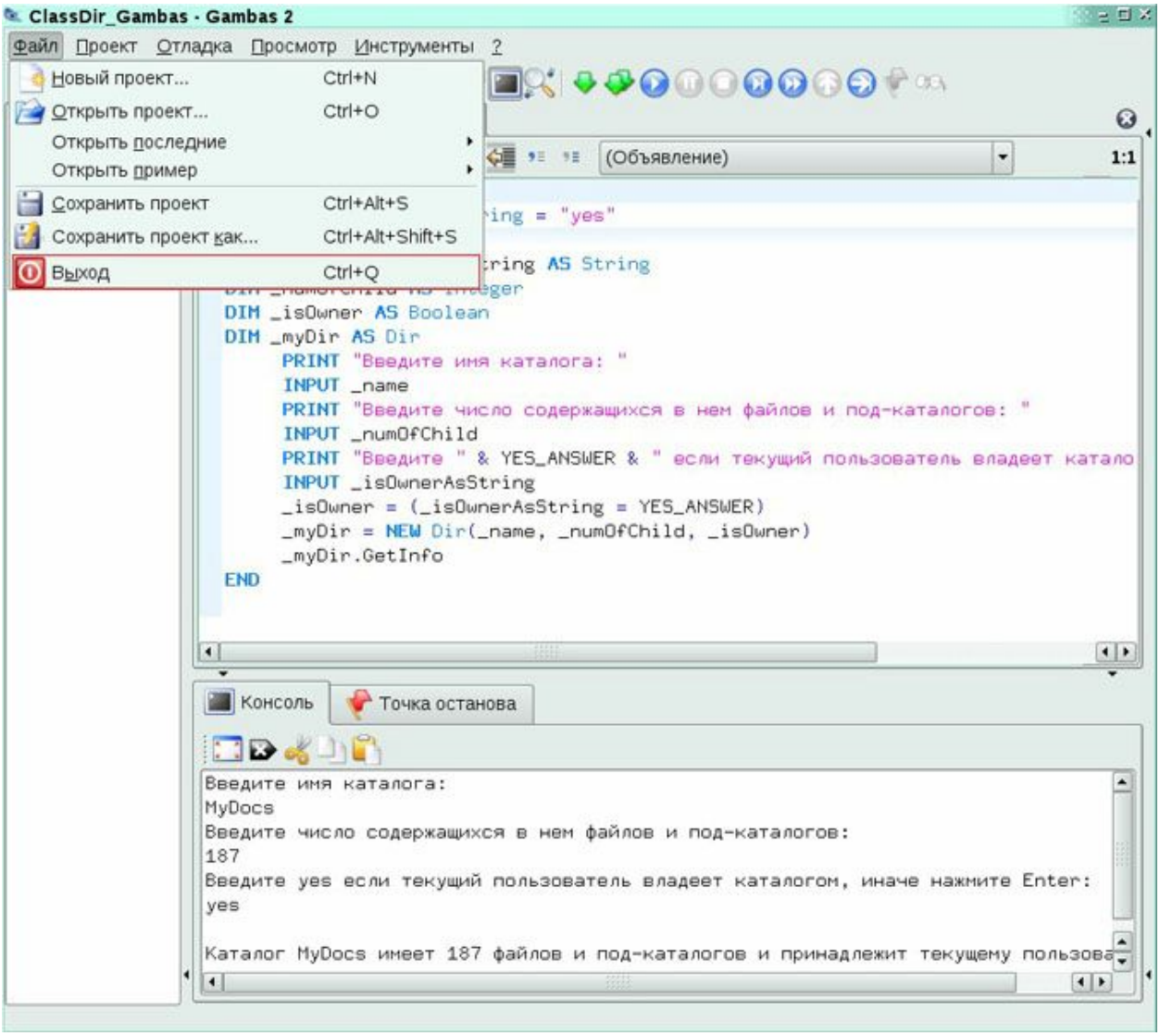

Рис. 72

29. Посмотрим, где на диске разместились файлы исходного кода, а так же созданный в последней операции архивный пакет этих файлов. Откройте универсальный браузер Konqueror и перейдите в наш домашний каталог (/home/admin). В списке содержащихся в нем подпапок можно видеть и корневую папку нашего проекта - **ClassDir_Gambas** (Рис. 73)

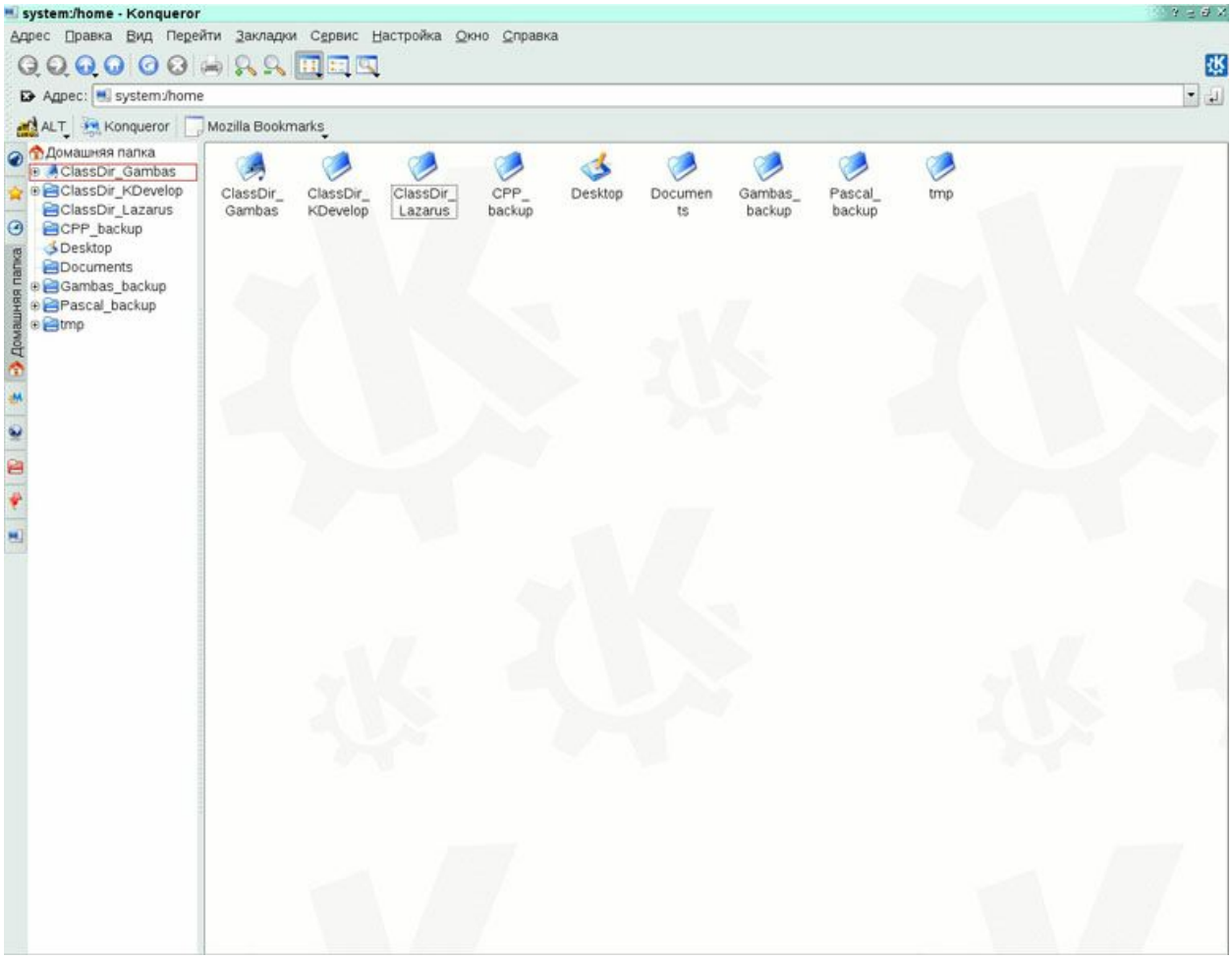

Рис. 73

30. В дереве папок слева выберите папку **ClassDir_Gambas**. Состав этой папки будет таким:

- файлы исходного кода входящие в проект - **MMain.module** и **Dir.class**
- резервные копии этих файлов (у них к расширению добавляется тильда, **~**)
- файл, содержащий компрессированную версию кода всех исходных файлов (архив или пакет) - **ClassDir_Gambas.gambas**. Он выделен на Рис. 74.

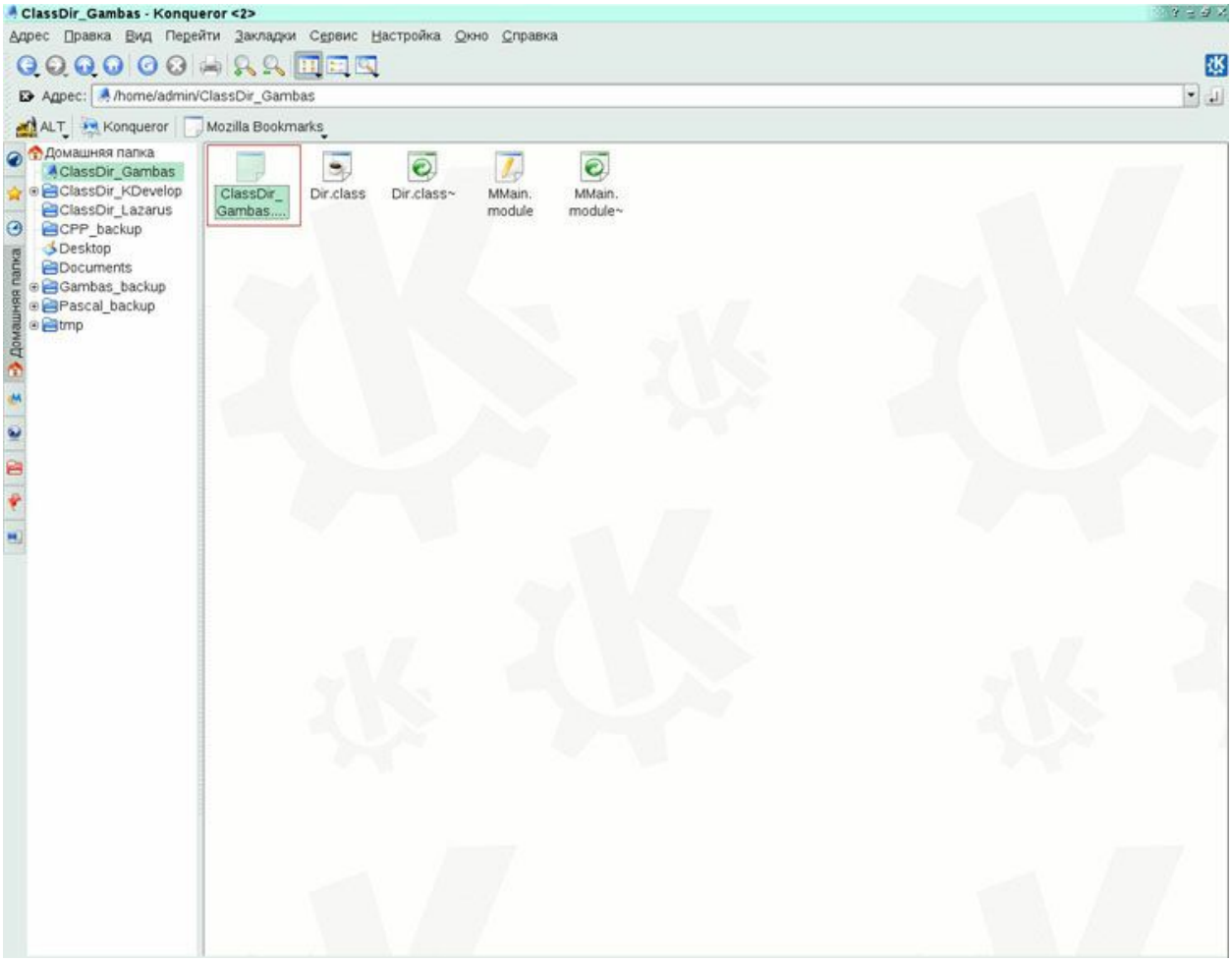

Рис. 74

31. Для распространения нашей готовой программы нам достаточно передать лишь один этот архив. Целевому компьютеру потребуется, как всегда, идентичная/совместимая версия ОС Linux и плюс интерпретатор языка Gambas. Последний может иметь имена **gbx2** или **gbr2**, в зависимости от того предназначен ли он для запуска непосредственно исходных файлов (.module/.class) или архива (.gambas). Поскольку мы передаем файл именно последнего типа и предположив что на целевом компьютере Gambas-интерпретатор корректно проинсталлирован и настроен мы можем в консоли целевого компьютера дать команду:

```
[admin@mlinux xxx]$ gbr2 ClassDir_Gambas.gambas
```

где *xxx* - папка целевого компьютера, в которую мы и скопировали выходной архив. Такая команда приведет к инициированию только что опробованному нами диалогу из трех вопросов и одного информационного сообщения.