Содержание

Задание на курсовую работу

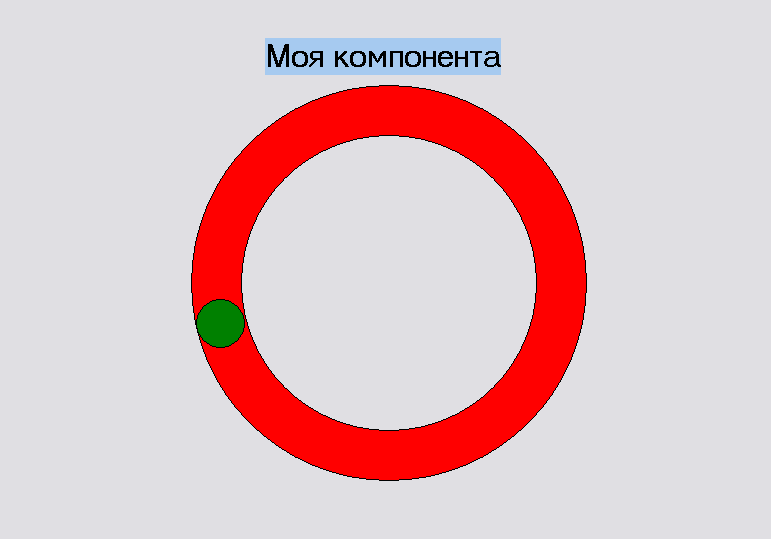
Введение

1. Разработка компоненты
2. Задание начальных условий
3. Задание свойств
4. Переопределённые методы базового класса
5. Переопределение метода CreateWnd()
6. Переопределение метода Paint()
7. Переопределение метода WndProc(TMessage& Message)
8. Переопределение метода CanResize(int &NewWidth, int &NewHeight)
9. Создание событий
10. Тестирование компоненты
11. Регистрация компоненты
12. Инсталляция компоненты
13. Проверка готовой компоненты

Список используемой литературы

**Задание на курсовую работу**

Разработать компоненту, состоящую из двух окружностей, одна из которых движется по периметру другой. Создать установочный пакет и тестовое приложение с использованием разработанного компонента.



**Введение**

C++ Builder представляет собой приложение, главное окно которого содержит настраиваемую инструментальную панель и палитру компонентов. Помимо этого, по умолчанию при запуске C++ Builder появляются окно инспектора объектов и форма нового приложения. Под окном формы приложения находится окно редактора кода.

Формы являются основой приложений C++ Builder. Создание пользовательского интерфейса приложения заключается в добавлении в окно формы элементов объектов C++ Builder, называемых компонентами. Компоненты C++ Builder располагаются на палитре компонентов, выполненной в виде многостраничного блокнота. Важная особенность C++ Builder состоит в том, что он позволяет создавать собственные компоненты и настраивать палитру компонентов, а также создавать различные версии палитры компонентов для разных проектов.

Компоненты разделяются на видимые (визуальные) и невидимые (невизуальные). Визуальные компоненты появляются во время выполнения точно так же, как и во время проектирования. Примерами являются кнопки и редактируемые поля. Невизуальные компоненты появляются во время проектирования как пиктограммы на форме. Они никогда не видны во время выполнения, но обладают определенной функциональностью (например, обеспечивают доступ к данным, вызывают стандартные диалоги Windows и др.)

Для добавления компонента в форму можно выбрать мышью нужный компонент в палитре и щелкнуть левой клавишей мыши в нужном месте проектируемой формы. Компонент появится на форме, и далее его можно перемещать, менять размеры и другие характеристики.

Каждый компонент C++ Builder имеет три разновидности характеристик: свойства, события и методы.

Если выбрать компонент из палитры и добавить его к форме, инспектор объектов автоматически покажет свойства и события, которые могут быть использованы с этим компонентом. В верхней части инспектора объектов имеется выпадающий список, позволяющий выбирать нужный объект из имеющихся на форме.

**1 Разработка компоненты**

Процесс разработки компоненты **TOsnova** проходит через выполнение следующих этапов:

1. Создание модуля для новой компоненты.

2. Наследование производного класса от существующего базового компонентного класса.

3. Добавление нужных свойств, событий и методов.

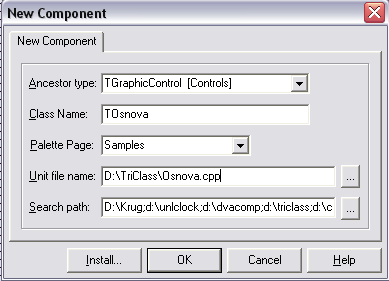
4. Регистрация компоненты в C++Builder.

Для создания нового компонента необходимо начать новый проект. Но сначала необходимо решить, от какого базового класса будет порождён наш компонент.

Мне видится, что наиболее правильным решением будет выбор в качестве базового класс TCustomControl. TCustomControl был выбран потому, что для полноценной работы нашей компоненте необходимы свойства Handle (дескриптор окна для получения сообщений таймера) и Canvas (холст для вывода текста). TCustomControl имеет оба свойства, за счет чего он, по сути, является идеальным выбором базового класса.

Также нам понадобится создать класс TMarshrut для рисования маршрута движения шарика и класс ТShar, рисующий сам шарик. Оба этих класса должны лишь уметь рисовать себя, поэтому в качестве базового класса выбираем TCustomGraphic.

Теперь можно приступать к созданию компоненты. Открываем вкладку Component и выбираем New Component. В появившемся окне выбираем базовый класс (Ancestor type) − TCustomControl, вводим имя своего класса (Class Name) − TOsnova, выбираем вкладку палитры компонентов, на которую компонента будет установлена (Palette page) −Samples, и путь, где будут хранится файлы компоненты. После щелчка на кнопке ОК C++ Builder создаст для нашей компоненты новый модуль и сохранит его в указанном месте.



Аналогичным образом создаём два класса TMarshrut и ТShar. Для совместной работы трёх классов необходимо подключить h−файлы с помощью директивы #include.

Теперь рассмотрим созданные нами классы. TMarshrut и ТShar должны уметь рисовать себя, поэтому в них достаточно переопределить виртуальный метод базового класса Paint():

virtual void \_\_fastcall Paint();

Переопределенный метод Paint() этих классов должен включать в себя код их рисования. Ниже показан исходный код метода Paint() класса ТShar:

void \_\_fastcall TShar::Paint()

{

Canvas->Brush->Color=clGreen;

Shar->Width=(D-d)/2;

Shar->Height=(D-d)/2;

Canvas->Ellipse(0,0,Shar->Width,Shar->Height);

}

Исходный код метода Paint() класса TMarshrut

//---------------------------------------------------------------------------

// рисуем две окружности заданного цвета

void \_\_fastcall TMarshrut::Paint()

{

Canvas->Brush->Color=clRed;

Marshrut->Ris\_Marshrut(D/2);

Canvas->Brush->Color=clBtnFace;

Marshrut->Ris\_Marshrut(d/2);

}

//---------------------------------------------------------------------------

Реализация функции Ris\_Marshrut(int R) (прототип находится в секции protected класса ТMarshrut):

//---------------------------------------------------------------------------

//рисуем окружность заданного радиуса

void \_\_fastcall TMarshrut::Ris\_Marshrut(int R)

{

Marshrut->Canvas->Ellipse(Marshrut->Width/2-R,Marshrut->Height/2-R,

Marshrut->Width/2+R,Marshrut->Height/2+R);

}

//---------------------------------------------------------------------------

Так как никаких действий (перемещение, изменение размеров и т.д.) не планируется, маршрут движения рисуется методом Paint() при помощи функции Ris\_Marshrut (int R). В противном случае лучше создать два экземпляра класса

ТMarshrut.

**2 Задание начальных условий**

Начальные значения задаются в конструкторе класса TOsnova.

\_\_fastcall TOsnova::TOsnova(TComponent\* Owner)

: TCustomControl(Owner)

{

Width=200;//начальная ширина

Height=200;//начальная высота

D=this->Width/5\*4;//диаметр большой окружности

d=this->Width/5\*3; //диаметр маленькой окружности

a=0;//угол поворота

N=true;//для однократного захода в цикл

FSkorosty=20;//скорость вращения

}

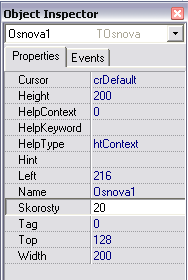
Начальные значения для экземпляров классов TMarshrut и ТShar задаются при их создании в функции void \_\_fastcall CreateWnd().

**3 Задание свойств**

В процессе работы желательно иметь возможность изменять скорость вращения шарика, поэтому добавим свойство FSkorosty. Тип этого свойства int. Изменение местоположения шарика будет происходить по сигналу таймера, поэтому, увеличивая или уменьшая значение интервала, можно менять скорость. Член данных FSkorosty размещён в секции private. Теперь надо объявить свойство − метод чтения и записи (в секции \_\_published)

\_\_property int Skorosty = { read=FSkorosty, write=SetSkorosty },

после чего это свойство отобразится в окне Инспектора Объектов.



Свойство Skorosty имеет прямой доступ к полю чтения, а для записи имеется метод SetSkorosty

//---------------------------------------------------------------------------

void \_\_fastcall TOsnova::SetSkorosty(int Skorosty)

{

FSkorosty=Skorosty;

if (FSkorosty<5) FSkorosty=5; // если FSkorosty=0, шарик остановится

SetTimer(Handle, 1, FSkorosty, 0);

}

//---------------------------------------------------------------------------

**4 Переопределённые методы базового класса**

Переопределённые методы базового класса объявлены в секции protected.

**5 Переопределение метода CreateWnd().**

При работе компоненты необходимо динамически создать экземпляры классов TMarshrut и ТShar. Хотелось бы сделать это непосредственно в конструкторе класса TOsnova, но попытка запуска окончится генерированием исключения времени выполнения:

Control has no parent window (Управляющий элемент не имеет родительского окна). Решение состоит в переопределении метода CreateWnd().

//---------------------------------------------------------------------------

void \_\_fastcall TOsnova::CreateWnd()

{

if (ComponentState.Contains(csDestroying))

return; //если компонента разрушается - выход

TCustomControl::CreateWnd(); //базовый метод

Marshrut=new TMarshrut(this); //создаём экземпляр класса

Marshrut->Parent=this;

Marshrut->Width=this->Width;

Marshrut->Height=this->Height;

Shar=new TShar(this); //создаём экземпляр класса

Shar->Parent=this;

Shar->Width=(D-d)/2;

Shar->Height=(D-d)/2;

Shar->Left=(this->Width-D)/2+D-Shar->Width;

Shar->Top=this->Height/2-Shar->Height/2;

if (ComponentState.Contains(csDesigning))

return; //если компонента разрабатывается - выход

else; //иначе запускаем таймер

SetTimer(Handle, 1, FSkorosty, 0);

}

//---------------------------------------------------------------------------

Сначала вызывается функция CreateWnd() базового класса, которая вернёт дескриптор окна. После этого мы можем создать экземпляры наших классов, а также функцией SetTimer() создать таймер.

**6 Переопределение метода Paint()**

//---------------------------------------------------------------------------

void \_\_fastcall TOsnova::Paint()

{

this->Canvas->Brush->Color=clSkyBlue;

this->Canvas->Font->Size=this->Height/20;

this->Canvas->TextOutA(this->Width/4,3,"Моя компонента");

}

//---------------------------------------------------------------------------

С помощью метода Paint() выведем текст на компоненту.

**7 Переопределение метода WndProc(TMessage& Message)**

Каждый компонент, имеющий свой собственный дескриптор окна, имеет и виртуальный метод по имени WndProc(). Этот метод вызывается каждый раз, когда Windows или VCL посылает сообщение окну компонента.

Прежде чем начать описание применения метода WndProc(), сосредоточимся на использовании таймера в компоненте. Таймер запускается в методе CreateWnd():

SetTimer (Handle, 1, FSkorosty, 0). В эту функцию в качестве параметров передаются дескриптор окна компонента, 1 − как идентификатор таймера, FSkorosty − как интервал таймера) и 0 в качестве значения последнего параметра, который используется только в случае, если вы применяете функцию обратного вызова таймера. В этом примере сообщается, что Windows следует посылать сообщение WM\_TIMER моей оконной процедуре окна.

Удаление таймера осуществляется в деструкторе компоненты:

KillTimer (Handle, 1) ;

Внутри этого метода выполняется соответствующая обработка сообщения

WM\_TIMER:

//---------------------------------------------------------------------------

void \_\_fastcall TOsnova::WndProc(TMessage& Message)

{

TCustomControl::WndProc(Message); //базовый метод

if (Message.Msg == WM\_TIMER)

{

//проверяем, изменились ли размеры

if ((N==true)&&( this->Width!=NewWidth)&&(this->Height!=NewHeight))

{

this->Width=NewWidth;

this->Height=NewHeight;

D=this->Width/5\*4;

d=this->Width/5\*3;

Marshrut->Width=this->Width;

Marshrut->Height=this->Height

Shar->Width=(D-d)/2;

Shar->Height=(D-d)/2;

Shar->Left=(this->Width-D)/2+D-Shar->Width;

Shar->Top=this->Height/2-Shar->Height/2;

this->Canvas->Font->Size=this->Height/20;

this->Canvas->TextOutA(this->Width/4,3,"Моя компонента");

this->Invalidate();//перерисовываем компоненту

N=false;//больше сюда не заходим

}

a+=0.2;//увеличиваем значение угла поворота (процесс движения)

Shar->Left=(Marshrut->Width-(D-(D-d)/2))/2+(D-(D-d)/2)/2+(D-(D-d)/2)/2\*cos(a)-

Shar->Width/2;

Shar->Top=(Marshrut->Height-(D-(D-d)/2))/2+(D-(D-d)/2)/2-(D-(D-d)/2)/2\*sin(a)-

Shar->Height/2;

}

}

//---------------------------------------------------------------------------

Сначала вызывается метод базового класса TCustomControl::WndProc(Message), затем устанавливаются новые размеры (если они были изменены), вызывается функция Invalidate() для перерисовки компоненты, и присваиваются новые координаты шарику.

**8 Переопределение метода CanResize(int &NewWidth, int &NewHeight).**

Если пользователь изменит размеры компоненты в режиме разработки, пропорционально должны измениться размеры всех объектов, из которых состоит компонента.

//---------------------------------------------------------------------------

bool \_\_fastcall TOsnova::CanResize(int &NewWidth, int &NewHeight)

{ //изменяем

if (ComponentState.Contains(csDesigning))

{

D=this->Width/5\*4; // диаметры окружностей

d=this->Width/5\*3;

Marshrut->Width=this->Width; //размеры экз. класса ТMarshrut

Marshrut->Height=this->Height;

Shar->Width=(D-d)/2; //размеры экз. класса ТShar

Shar->Height=(D-d)/2;

Shar->Left=(this->Width-D)/2+D-Shar->Width;

Shar->Top=this->Height/2-Shar->Height/2;

this->Canvas->Font->Size=this->Height/20; //размер шрифта

this->Canvas->TextOutA(this->Width/4,3,"Моя компонента"); //координаты

this->Invalidate(); //перерисовываем компоненту

}

if (NewWidth!=NewHeight) NewWidth=NewHeight;

return(NewWidth, NewHeight);

}

//---------------------------------------------------------------------------

Метод CanResize() возвращает новые значения ширины (NewWidth) и высоты (NewHeight), которые затем будут переданы методу WndProc(TMessage& Message), для того, чтобы компонента рисовалась в приложении с новыми размерами.

**9 Создание событий**

Наиболее простая форма события — это событие, просто уведомляющее о том, что оно произошло. Например, событие OnClick, которое присутствует в большинстве визуальных компонентов, имеет только один параметр — Sender:

void fastcall TForml::FormClick(TObject \*Sender).

Этот тип события определяется в VCL как TNotifyEvent.

При создании события, обеспечивающего простое уведомление, можно

воспользоваться типом события TNotifyEvent. OnMyClick, событие TOsnova — простое событие уведомления. Оно объявляется следующим образом:

\_\_property TNotifyEvent OnMyClick = {read=FOnMyClick, write=FOnMyClick};

Каждое событие должно иметь виртуальный метод, который используется внутри компоненты, чтобы сообщить ей о своем возникновении. Такие методы объявляются виртуальными (чтобы классы-наследники могли получить к ним доступ) и в защищенном разделе.

Виртуальный метод, сообщающий о возникновении события OnMyClick, называется DoOnMyClick():

//---------------------------------------------------------------------------

void \_\_fastcall TOsnova::DoOnMyClick()

{

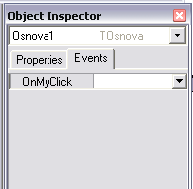
if (FOnMyClick)

FOnMyClick(this);

}

//---------------------------------------------------------------------------

Сначала проверяется, был ли событию назначен обработчик. Если обработчик события был назначен, то вызывается функция, указатель на которую расположен в поле FOnMyClick, куда передается в качестве параметра указатель на компонент (this). Это событие будет отражено на вкладке Events в окне Инспектора Объектов.



При двойном клике Builder сгенерирует следующий код:

//---------------------------------------------------------------------------

void \_\_fastcall TForm1::Osnova1MyClick(TObject \*Sender)

{

//код обработчика события

}

//---------------------------------------------------------------------------

**10 Тестирование компоненты**

Для тестирования компоненты начнём новый проект (File→New→Application).

Добавим к проекту (Project→Add to Project) файлы Osnova.cpp, Marshrut.cpp и Shar.cpp. В конструкторе формы создадим объект нашего класса

//---------------------------------------------------------------------------

\_\_fastcall TForm1::TForm1(TComponent\* Owner)

: TForm(Owner)

{

TOsnva \*Osnova;

Osnova=new TOsnova(this);

Osnova->Parent=this;

}

//---------------------------------------------------------------------------

Теперь все изменения в компоненте мы сразу сможем увидеть при запуске этого проекта.

**11 Регистрация компоненты**

Компонента регистрируется функцией Register(), которая автоматически генерируется C++ Builder. Ключевое слово namespace устанавливает локальность имён данной процедуры регистрации. После этого ключевого слова следует имя файла, содержащего компоненту. В процедуре регистрации Register первый оператор создаёт массив регистрируемых компонентов classes типа TComponentClass и заносит его в регистрируемую компоненту. Следующий оператор регистрирует функцией RegisterComponents компоненты, занесённые в classes (второй параметр функции) на странице Samples (второй параметр).

//---------------------------------------------------------------------------

namespace Osnova

{

void \_\_fastcall PACKAGE Register()

{

TComponentClass classes[1] = {\_\_classid(TOsnova)};

RegisterComponents("Samples", classes, 0);

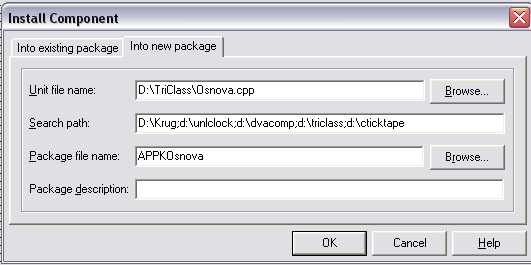
}

}

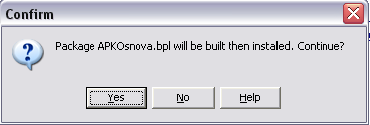
//---------------------------------------------------------------------------

**12 Инсталляция компоненты**

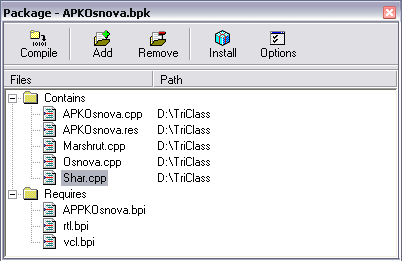
Для инсталляции компоненты выбираем меню Component, Install Component.



В появившемся окне выбираем вкладку Into new package (в новый пакет), вводим имя пакета (APPKOsnova) и нажимаем ОК. Появляется окошко с предложением инсталлировать пакет.

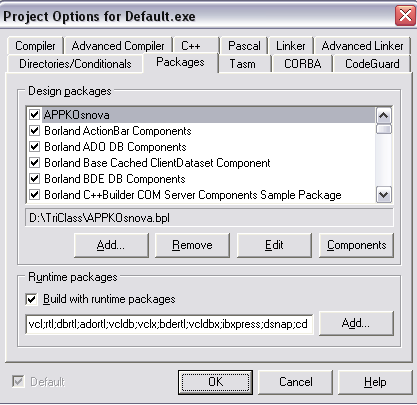


Отвечаем No, так как наш пакет полностью ещё не собран − в него не включены классы TMarshrut и ТShar. В окне Диспетчера Пакетов (Package Manager) выбираем Add, Add Unit и добавляем файлы Marshrut.cpp и Shar.cpp



После компиляции можно инсталлировать нашу компоненту.

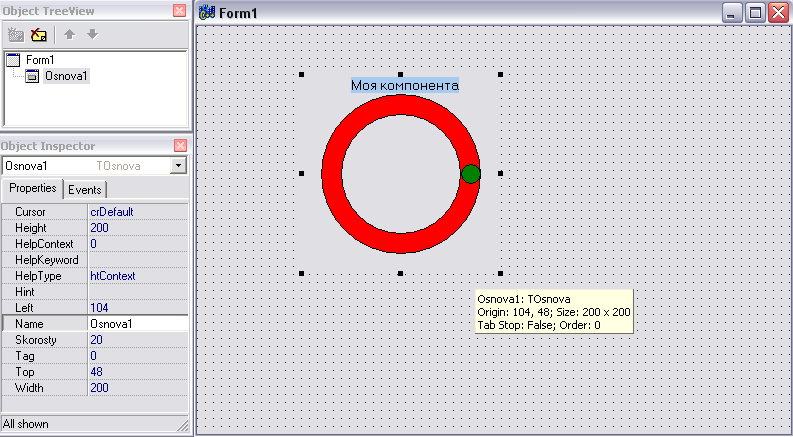
Деинсталлировать компоненту можно, выбрав вкладку Project −> Options −>Remove.



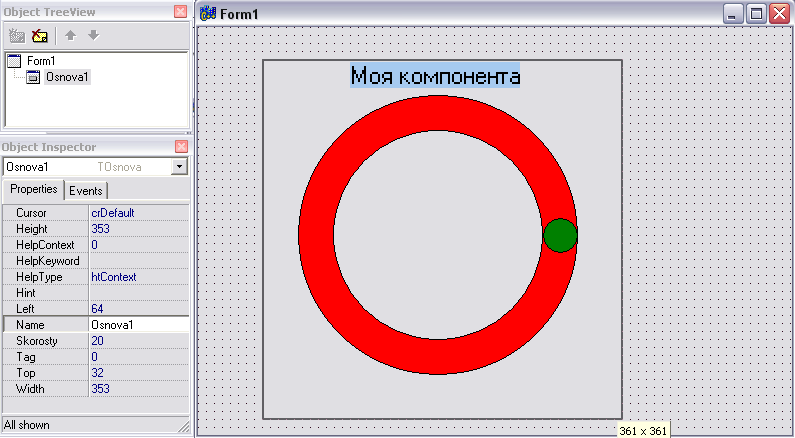
**13 Проверка готовой компоненты**

Для проверки работоспособности компоненты начнём новый проект (File→New→Application).

Выбираем на палитре компонентов вкладку Samples и переносим нашу компоненту на форму.



Потянув за рамку, мы можем менять размеры компоненты



Добавим на форму две кнопки и в обработчике события OnClick напишем код

//---------------------------------------------------------------------------

void \_\_fastcall TForm1::Scororty\_plusClick(TObject \*Sender)

{

Form1->Osnova1->Skorosty-=5;

}

//---------------------------------------------------------------------------

void \_\_fastcall TForm1::Scororty\_minusClick(TObject \*Sender)

{

Form1->Osnova1->Skorosty+=5;

}

//---------------------------------------------------------------------------

и запустим проект. При каждом нажатии на кнопку Scororty\_plus скорость вращения будет увеличиваться, а при нажатии на кнопку Scororty\_minus − уменьшаться.

**Список используемой литературы**

1. А.Я. Архангельский «Программирование в С++ Builder 6».

2. К.Рейсдорф, К.Хендерсон «Borland C++ Builder. Освой самостоятельно

за 21 день».

3. Теллес М. « Borland C++ Builder. Библиотека программиста(1998)»

4. Ч. Калверт, К. Рейсдорф «Borland C++ Builder».