Приднестровский государственный университет им. Т.Г. Шевченко

Рыбницкий филиал

*Кафедра физики, математики и информатики*

**КУРСОВАЯ РАБОТА**

**по дисциплине**

**«Объектно-ориентированное программирование»**

**на тему:**

**«Реализация объекта “круг ” на языке программирования C++»**

студента II курса

направления «Программная инженерия»

профиля «Разработка программно–информационных систем»

Толстова И. И.

Научный руководитель:

преподаватель

Нагаевский О. М.

Рыбница, 2015

Оглавление

[ВВЕДЕНИЕ 3](#_Toc421238623)

[Глава I. Исследование и анализ предметной области 4](#_Toc421238624)

[1.1 Принципы представления изображения. 4](#_Toc421238625)

[1.2 Растровая графика и пиксель 8](#_Toc421238626)

[1.3 Векторная графика 11](#_Toc421238627)

[1.4 Преимущества и недостатки растровой и векторной графики 12](#_Toc421238628)

[1.5 Сравнение растровых и векторных изображений 15](#_Toc421238629)

[ГЛАВА II. РЕАЛИЗАЦИЯ И ТЕСТИРОВАНИЕ ПРОГРАММНОГО ПРОДУКТА 17](#_Toc421238630)

[2.1 Классы 17](#_Toc421238631)

[2.2 Среда разработки 20](#_Toc421238632)

[2.3 Реализация программного продукта 21](#_Toc421238633)

[2.4 Пользовательский интерфейс 22](#_Toc421238634)

[2.5 Тестирование программного продукта 22](#_Toc421238635)

[Заключение 24](#_Toc421238636)

[Список литературы 25](#_Toc421238637)

[ЛИСТИНГ 26](#_Toc421238638)

ВВЕДЕНИЕ

С развитием технологий компьютеры все больше и больше вливаются в повседневную жизнь человека. Персональные компьютеры – это уже не просто машины для математических вычислений, их общедоступность значительно расширила область применения. И, конечно же, здесь не обошлось без игр. Индустрия игр быстро набирала обороты, стремительно развивалась, завоевывая все большую и большую популярность у подрастающего поколения. И если первые игры отличались простотой дизайна и логики, то современная компьютерная игра – очень сложная программа, дело рук большого коллектива разработчиков. Так или иначе, компьютерные игры занимают не последнее место в жизни современных людей.

Актуальность исследования определяется тем, что реализация объектов – это одно из самых интересных направлений в объектно-ориентированном программировании. Оно позволяет изучить графические возможности компьютера, глубже понять принцип его работы.

Целью данной курсовой работы является реализация объекта круг при помощи средств языка программирования C++ Builder. Для достижения данной цели были поставлены следующие задачи:

1. Изучить теоретические сведения, необходимые для решения данной задачи.
2. Систематизировать и обобщить полученные знания.
3. Создать программный продукт, реализующий объект круг.
4. Провести тестирование и отладку программы.

Цель и задачи курсовой работы определили её структуру. Она состоит из введения, двух разделов, заключения, списка литературы и приложения.

# Глава I. Исследование и анализ предметной области

## 1.1 Принципы представления изображения.

Компьютерная графика прочно вошла в нашу жизнь. Появляется все больше клипов, сделанных с помощью компьютерной графики. Нет спору, компьютерная графика расширяет выразительные возможности. При творческом ее использовании реклама приобретает удивительную силу воздействия на зрителя. С помощью одной только компьютерной графики очень трудно донести до зрителя рекламную идею. И если в клипе лишь компьютерная графика, лишь созданный ее средствами сюрреалистический мир, то зритель остается холодным, хотя увиденное и поражает воображение. Ведь известно, что реклама наиболее эффективна тогда, когда потребителю хочется идентифицировать себя с человеком, пользующимся тем или иным товаром. Процесс узнаваемости себя в клипе - залог успеха.

Компьютерная или машинная графика - это вполне самостоятельная область человеческой деятельности, со своими проблемами и спецификой. Компьютерная графика - это и новые эффективные технические средства для проектировщиков, конструкторов и исследователей, и программные системы и машинные языки, и новые научные, учебные дисциплины, родившиеся на базе синтеза таких наук как аналитическая, прикладная и начертательная геометрии, программирование для ПК, методы вычислительной математики и т.п. Машина наглядно изображает такие сложные геометрические объекты, которые раньше математики даже не пытались изобразить.

Реализация объекта круг, по своей сути, подразумевает реализацию упрощенного векторного редактора, в котором можно выполнять различные действия с данным объектом, такие как: вращение, перемещение, отражение, растягивание и т.д. В данной главе нельзя описать историю возникновения конкретной фигуры и всех остальных фигур по причине того, что это описание почти эквивалентно описанию того, как появилась жизнь на Земле. Поэтому здесь будет представлена история возникновения редакторов графической информации и общая информация о них.

В компьютерной графике существуют два различных подхода к представлению графической информации**.** Они называются растровым и векторным, соответственно. Суть растрового подхода в том, что всякое изображение рассматривается как совокупность точек разного цвета. Векторный подход рассматривает изображение как совокупность простых элементов: прямых линий, дуг, окружностей, эллипсов, прямоугольников, закрасок и пр., которые называются графическими примитивами.

В растровой графике графическая информация — это совокупность данных о цветах пиксельей на экране. В векторной графике графическая информация — это данные, однозначно определяющие все графические примитивы, составляющие рисунок.

Положение и форма графических примитивов задаются в системе графических координат, связанных с экраном. Обычно начало координат расположено в верхнем левом углу экрана. Сетка пиксельей совпадает с координатной сеткой. Горизонтальная ось X направлена слева направо; вертикальная ось Y — сверху вниз.

Отрезок прямой линии однозначно определяется указанием координат его концов; окружность — координатами центра и радиусом; многоугольник — координатами его вершин; закрашенная область — граничной линией и цветом закраски и пр.

**Графический редактор** — программа (или пакет программ), позволяющая создавать, просматривать, обрабатывать и редактировать цифровые изображения (рисунки, картинки, фотографии) на компьютере. Типы графических редакторов выглядят следующим образом:

* Растровые графические редакторы. Наиболее популярные профессиональные растровые графические редакторы: платный **Adobe Photoshop** (для операционных систем Windows и Windows Phone, iOS, Android) и его бесплатный аналог **GIMP** (для операционных систем Linux и Windows, Free BCD и Solaris), распространяемый под лицензией GNU GPL, а также менее известные графические редакторы для начинающих: любительский **Photofiltre** и учебный **Paint.NET.**
* Векторные графические редакторы. Наиболее популярные профессиональные векторные графические редакторы: платный **Adobe Illustrator** и платный **Corel Draw (**для операционных систем Microsoft Windows и Mac OS X), а также свободно распространяемый и бесплатный графический редактор **Inkscape** — для всех ОС.
* Гибридные графические редакторы. Наиболее популярны: **RasterDesk** для **AutoCAD**, **Spotligh**t для операционных систем Microsoft Windows. Редакторы растровой графики **Adobe Photoshop, Adobe Illustrator** и **Corel Draw** также поддерживают некоторые функции для работы над векторной графикой.

**Векторная графика** — способ представления объектов и изображений в компьютерной графике, основанная на математическом описании элементарных геометрических объектов, таких как: точки, линии, сплайны, кривые Безье и многоугольники. Объекты векторной графики являются графическими изображениями математических объектов.

Термин используется для различения от растровой графики, в которой изображение представлено в виде графической матрицы, состоящей из пикселей фиксированного размера. Каждому пикселю графической матрицы в растровом изображении приписан атрибут цвета. Совокупность разноцветных пикселей растровой матрицы формирует изображение.

Основной недостаток растровой графики — большой размер графических файлов. Простые растровые картинки занимают несколько десятков или сотен килобайтов. Реалистические изображения, полученные с помощью сканеров с высокой разрешающей способностью, могут занимать несколько мегабайтов. По этой причине информация в файлах растрового формата, как правило, хранится в сжатом виде. Для сжатия графической информации используются специальные методы, позволяющие сократить ее объем в десятки раз.

Еще одним недостатком растровых изображений является их искажение, возникающее при изменении размеров, вращении и других преобразованиях. Картинка, которая прекрасно выглядела при одном размере, после масштабирования или вращения может потерять свою привлекательность. Например, в областях однотонной закраски могут появиться ненужные узоры; кривые и прямые линии могут приобрести пилообразную форму и т. п.

Векторные изображения получаются с помощью графических редакторов векторного типа (их еще называют пакетами иллюстративной графики). Эти пакеты предоставляют в распоряжение пользователя набор инструментов и команд, с помощью которых создаются рисунки. Прямые линии, окружности, эллипсы и дуги являются основными компонентами векторных изображений. Одновременно с процессом рисования специальное программное обеспечение формирует описания графических примитивов, из которых строится рисунок. Эти описания сохраняются в графическом файле.

Графические файлы векторного типа имеют относительно небольшие размеры, Рисунки, состоящие из тысяч примитивов, занимают дисковую память, объем которой не превышает нескольких сотен килобайтов. Аналогичный растровый рисунок требует в 10-1000 раз большую память.

Векторные изображения легко масштабируются без потери качества, Например, для увеличения или уменьшения эллипса достаточно изменить координаты левого верхнего и правого нижнего углов прямоугольника, ограничивающего этот эллипс.

Следует понимать, что различие в представлении графической информации существует лишь для файлов. При выводе на экран любого изображения в видео формате в памяти формируется информация, содержащая данные о цвете каждого пикселья экрана.

**1.2 Растровая графика и пиксель**

Первые вычислительные машины не имели отдельных средств для работы с графикой, однако уже использовались для получения и обработки изображений. Программируя память первых электронных машин, построенную на основе матрицы ламп, можно было получать узористые картины.

В 1961 году программист С. Рассел возглавил проект по созданию первой компьютерной игры с графикой. Создание игры «Spacewar» («Космические войны») заняло около 200 человеко-часов. Игра была создана на машине PDP-1.

В 1963 году американский учёный Айвен Сазерленд создал программно-аппаратный комплекс Sketchpad, который позволял рисовать точки, линии и окружности на трубке цифровым пером. Поддерживались базовые действия с примитивами: перемещение, копирование и др. По сути, это был первый растровый редактор, реализованный на компьютере. Также программу можно назвать первым графическим интерфейсом, причём она являлась таковой ещё до появления самого термина.

В середине 1960-х гг. появились разработки в промышленных приложениях компьютерной графики. Так, под руководством Т. Мофетта и Н. Тейлора фирма Itek разработала цифровую электронную чертёжную машину. В 1964 году General Motors представила систему автоматизированного проектирования DAC-1, разработанную совместно с IBM.

В 1968 году группой под руководством Константинова Н. Н. была создана компьютерная математическая модель движения кошки. Машина БЭСМ-4, выполняя написанную программу решения дифференциальных уравнений, рисовала мультфильм «Кошечка», который для своего времени являлся прорывом. Для визуализации использовался алфавитно-цифровой принтер. Существенный прогресс компьютерная графика испытала с появлением возможности запоминать изображения и выводить их на компьютерном дисплее.

Компьютерная индустрия породила сотни новых и необычных терминов, пытаясь объяснить, что такое компьютер и как он работает. Термин растровая графика достаточно очевиден, если усвоить понятия, относящиеся к растровым изображениям. Растровые изображения напоминают лист клетчатой бумаги, на котором любая клетка закрашена либо черным, либо белым цветом, образуя в совокупности рисунок.

Пиксель – основной элемент растровых изображений. Именно из таких элементов состоит растровое изображение. В цифровом мире компьютерных изображений термином пиксель обозначают несколько разных понятий. Это может быть отдельная точка экрана компьютера, отдельная точка напечатанная на лазерном принтере или отдельный элемент растрового изображения. Эти понятия не одно и тоже, поэтому чтобы избежать путаницы следует называть их следующим образом: видео пиксель при ссылке на изображение экрана компьютера; точка при ссылке на отдельную точку, создаваемую лазерным принтером. Существует коэффициент прямоугольности изображения, который введен специально для изображения количества пикселей матрицы рисунка по горизонтали и по вертикали.

Возвращаясь к аналогии с листом бумаги можно заметить, что любой растровый рисунок имеет определенное количество пикселей в горизонтальных и вертикальных рядах. Существуют следующие коэффициенты прямоугольности для экранов: 320х200, 320х240, 600х400, 640х480, 800х600 и др. Этот коэффициент часто называют размером изображения. Произведение этих двух чисел дает общее количество пикселей изображения.

Существует такое понятие как коэффициент прямоугольности пикселей. В отличие от коэффициента прямоугольности изображения он относится к реальным размерам видео пикселя и является отношением реальной ширины к реальной высоте. Данный коэффициент зависит от размера дисплея и текущего разрешения, и поэтому на разных компьютерных системах принимает различные значения. Цвет любого пикселя растрового изображения запоминается в компьютере с помощью комбинации битов. Чем больше битов для этого используется, тем больше оттенков цветов можно получить. Число битов, используемых компьютером для любого пикселя, называется битовой глубиной пикселя.

Наиболее простое растровое изображение состоит из пикселей имеющих только два возможных цвета: черный и белый, и поэтому изображения, состоящие из пикселей этого вида, называются однобитовыми изображениями. Число доступных цветов или градаций серого цвета равно 2 в степени равной количеству битов в пикселе.

Цвета, описываемые 24 битами, обеспечивают более 16 миллионов доступных цветов и их часто называют естественными цветами. Растровые изображения обладают множеством характеристик, которые должны быть организованы и фиксированы компьютером. Размеры изображения и расположение пикселей в нем это две основных характеристики, которые файл растровых изображений должен сохранить, чтобы создать картинку. Даже если испорчена информация о цвете любого пикселя и любых других характеристиках компьютер все равно сможет воссоздать версию рисунка, если будет знать, как расположены все его пиксели. Пиксель сам по себе не обладает никаким размером, он всего лишь область памяти компьютера, хранящая информацию о цвете, поэтому коэффициент прямоугольности изображения не соответствует никакой реальной размерности. Зная только коэффициент прямоугольности изображения с некоторой разрешающей способностью можно определить настоящие размеры рисунка.

Поскольку размеры изображения хранятся отдельно, пиксели запоминаются один за другим, как обычный блок данных. Компьютеру не приходится сохранять отдельные позиции, он всего лишь создает сетку по размерам заданным коэффициентом прямоугольности изображения, а затем заполняет ее пиксель за пикселем. Это самый простой способ хранения данного растрового изображения, но не самый эффективный с точки зрения использования компьютерного времени и памяти.

Более эффективный способ состоит в том, чтобы сохранить только количество черных и белых пикселей в любой строке. Этот метод сжимает данные, которые используют растровые изображения. В этом случае они занимают меньше памяти компьютера.

**1.3 Векторная графика**

**Векторная графика** — способ представления объектов и изображений в компьютерной графике, основанная на математическом описании элементарных геометрических объектов, таких как: точки, линии, сплайны, кривые Безье и многоугольники. Объекты векторной графики являются графическими изображениями математических объектов.

Векторные графические редакторы, типично, позволяют вращать, перемещать, отражать, растягивать, скашивать, выполнять основные аффинные преобразования над объектами, изменять z-order и комбинировать примитивы в более сложные объекты. Более изощрённые преобразования включают булевы операции на замкнутых фигурах: объединение, дополнение, пересечение и т.д.

Векторная графика идеальна для простых или составных рисунков, которые должны быть аппаратно-независимыми или не нуждаются в фотореализме. К примеру, PostScript и PDF используют модель векторной графики.

Основные инструменты графических редакторов:

Кривые Безье — позволяют создавать прямые, ломаные и гладкие кривые, проходящие через узловые точки, с определёнными касательными в этих точках;

* Заливка — позволяет закрашивать ограниченные области определённым цветом или градиентом;
* Текст создаётся с помощью соответствующего инструмента, а потом часто преобразуется в кривые, чтобы обеспечить независимость изображения от шрифтов, имеющихся (или отсутствующих) на компьютере, используемом для просмотра;
* Набор геометрических примитивов;
* Карандаш — позволяет создавать линии «от руки». При создании таких линий возникает большое количество узловых точек, от которых в дальнейшем можно избавиться с помощью «упрощения кривой».

**1.4 Преимущества и недостатки растровой и векторной графики**

Растровая графика эффективно представляет реальные образы. Реальный мир состоит из миллиардов мельчайших объектов и человеческий глаз как раз приспособлен для восприятия огромного набора дискретных элементов, образующих предметы.

На своем высшем уровне качества - изображение выглядят вполне реально подобно тому, как выглядят фотографии в сравнении с рисунками. Это верно только для очень детализированных изображений, обычно получаемых сканированием фотографий. Помимо естественного вида растровые изображения имеют другие преимущества. Устройства вывода, такие как лазерные принтеры, для создания изображений используют наборы точек.

Растровые изображения могут быть очень легко распечатаны на таких принтерах, потому что компьютерам легко управлять устройством вывода для представления отдельных пикселей с помощью точек.

Перечень преимуществ растровой графики:

* Растровая графика позволяет создать практически любой рисунок, вне зависимости от сложности, в отличие, например, от векторной, где невозможно точно передать эффект перехода от одного цвета к другому без потерь в размере файла;
* Распространённость — растровая графика используется сейчас практически везде: от маленьких значков до плакатов;
* Высокая скорость обработки сложных изображений, если не нужно масштабирование;
* Растровое представление изображения естественно для большинства устройств ввода-вывода графической информации, таких как мониторы (за исключением векторных устройств вывода), матричные и струйные принтеры, цифровые фотоаппараты, сканеры, а также сотовые телефоны.

Как уже говорилось, растровые изображения занимают большое количество памяти. Существует так же проблема редактирования растровых изображений, так как большие растровые изображения занимают значительные массивы памяти, то для обеспечения работы функций редактирования таких изображений потребляются так же значительные массивы памяти и другие ресурсы компьютера.

Перечень недостатков растровой графики:

* Большой размер файлов у простых изображений;
* Невозможность идеального масштабирования;
* Невозможность вывода на печать на векторный графопостроитель.

Из-за этих недостатков для хранения простых рисунков рекомендуют вместо даже сжатой растровой графики использовать векторную графику.

Перечень преимуществ векторной графики перед растровой:

* Размер, занимаемый описательной частью, не зависит от реальной величины объекта, что позволяет, используя минимальное количество информации, описать сколь угодно большой объект файлом минимального размера.
* В связи с тем, что информация об объекте хранится в описательной форме, можно бесконечно увеличить графический примитив, например, дугу окружности, и она останется гладкой. С другой стороны, если кривая представлена в виде ломаной линии, увеличение покажет, что она на самом деле не кривая.
* Параметры объектов хранятся и могут быть легко изменены. Также это означает что перемещение, масштабирование, вращение, заполнение и т.д. не ухудшает качества рисунка. Более того, обычно указывают размеры в аппаратно-независимых единицах (англ. *device-independent unit*), которые ведут к наилучшей возможной растеризации на растровых устройствах.
* При увеличении или уменьшении объектов толщина линий может быть задана постоянной величиной, независимо от реального контура.

Фундаментальные недостатки векторной графики:

* Не каждый объект может быть легко изображен в векторном виде — для подобного оригинальному изображению может потребоваться очень большое количество объектов с высокой сложностью, что негативно влияет на количество памяти, занимаемой изображением, и на время для его отображения (отрисовки).
* Перевод векторной графики в растр достаточно прост. Но обратного пути, как правило, нет — трассировка растра, при том что требует значительных вычислительных мощностей и времени, не всегда обеспечивает высокое качество векторного рисунка.
* При этом спецификации векторных форматов (и, соответственно, рендереры векторной графики) намного сложнее таковых для растровой графики.
* Преимущество векторной картинки — масштабируемость — ропадает, когда начинаем иметь дело с особо малыми разрешениями графики (например, иконки 32×32 или 16×16). Чтобы не было «грязи», картинку под такие разрешения приходится подгонять вручную. В векторных шрифтах TrueType есть довольно сложные коды хинтинга, позволяющие избавиться от пропущенных (и, наоборот, излишне толстых) линий.

**1.5 Сравнение растровых и векторных изображений**

Растр – формат представления изображения в виде элементов (пикселей) упорядоченных в строки и столбцы. Чем больше элементов в составе мозаики, тем однороднее выглядит изображение. Растровые изображения тоже характеризуются количеством составляющих их точек — пикселей.

Векторное изображение – это совокупность геометрических фигур, которые задаются математическими формулами. Например, точка в вектор­ном рисунке будет представлена вектором единичной длины. Для то­го, чтобы на­ри­со­вать пря­мую, до­ста­точ­но знать ко­ор­ди­на­ты двух ее то­чек, а для со­зда­ния кру­га не­об­хо­ди­мы ра­ди­ус и ко­ор­ди­на­ты его цен­тра. Эл­липс по­тре­бу­ет ко­ор­ди­на­ты цен­тра и зна­че­ния двух ра­ди­у­сов.



Рисунок 1 Сравнение 1

Из при­ме­ра хо­ро­шо вид­но, что раст­ро­вое изоб­ра­же­ние при уве­ли­че­нии «рас­па­да­ет­ся» на от­дель­ные пик­се­ли, в то вре­мя как век­тор явно выигрывает, со­хра­ня­я чет­кость.

И на­ко­нец, еще од­но важ­ное пре­иму­ще­ство век­то­ра: раз­ре­ше­ние век­тор­но­го ри­сун­ка за­ви­сит от раз­ре­ше­ния вы­вод­но­го устрой­ства и пе­ча­та­ет­ся с мак­си­маль­ным ка­че­ством, до­ступ­ным для дан­но­го устрой­ства.

Ло­го­ти­пы, ил­лю­стра­ции к бро­шю­рам и бук­ле­там, ви­зит­ки и пла­ка­ты, на­клей­ки и эти­кет­ки — эти и мно­гие дру­гие эле­мен­ты кор­по­ра­тив­но­го сти­ля и ре­кла­мы тре­бу­ют ис­пол­не­ния в век­тор­ном фор­ма­те. Ведь за­да­ча раз­ра­бот­чи­ков при их со­зда­нии — чет­кий ри­су­нок, лег­ко мас­шта­би­ру­е­мый и адап­ти­ру­е­мый для лю­бой про­дук­ции.



Второй пример, показывающий разницу между растровой и векторной графикой при увеличении. Растровые изображения плохо масштабируются, тогда как векторные могут быть неограниченно **Рисунок 2 Сравнение 2**

увеличены без потери качества (изображения были сконвертированы в SVG)

ГЛАВА II. РЕАЛИЗАЦИЯ И ТЕСТИРОВАНИЕ ПРОГРАММНОГО ПРОДУКТА

## 2.1 Классы

Класс – разновидность абстрактного типа данных в объектно–ориентированном программировании (ООП), характеризуемый способом своего построения. Другие абстрактные типы данных — метаклассы, интерфейсы,  структуры, перечисления, — характеризуются какими-то своими, другими особенностями. Наряду с понятием «объекта» класс является ключевым понятием в ООП (хотя существуют и бесклассовые объектно-ориентированные языки, например, Self, Lua.). Суть отличия классов от других абстрактных типов данных состоит в том, что при задании типа данных класс определяет одновременно и интерфейс, и реализацию для всех своих экземпляров, а вызов метода-конструктора обязателен.

В объектно-ориентированной программе с применением классов каждый объект является «*экземпляром*» некоторого конкретного класса, и других объектов не предусмотрено. То есть «*экземпляр класса*» в данном случае означает не «пример некоторого класса» или «отдельно взятый класс», а «объект, типом которого является какой-то класс». При этом в разных языках программирования допускается либо не допускается существование еще каких-то типов данных, экземпляры которых не являются объектами (то есть язык определяет, являются ли объектами такие вещи, как числа, массивы и указатели, или не являются, и, соответственно, есть ли такие классы как «число», «массив» или «указатель», экземплярами которых были бы каждое конкретное число, массив или указатель).

Например, абстрактный тип данных «строка текста» может быть оформлен в виде класса, и тогда все строки текста в программе будут являться объектами — экземплярами класса «строка текста». При использовании классов все элементы кода программы, такие как переменные, константы, методы, процедуры и функции, могут принадлежать (а во многих языках обязаны принадлежать) тому или иному классу. Сам класс в итоге определяется как список своих *членов*, а именно полей (свойств) и методов/функций/процедур. В зависимости от языка программирования к этому списку могут добавиться константы, атрибуты и внешние определения.

Как и структуры, классы могут задавать поля — то есть переменные, принадлежащие либо непосредственно самому классу (статические), либо экземплярам класса (обычные). Статические поля существуют в одном экземпляре на всю программу (или, в более сложном варианте, — в одном экземпляре на процесс или на поток/нить). Обычные поля создаются по одной копии для каждого конкретного объекта — экземпляра класса. Например, общее количество строк текста, созданных в программе за время её работы, будет являться статическим полем класса «строка текста». А конкретный массив символов строки будет являться обычным полем экземпляра класса «строка текста», так же как переменная «фамилия», имеющая тип «строка текста», будет являться обычным полем каждого конкретного экземпляра класса «человек».

В ООП при использовании классов весь исполняемый код программы (алгоритмы) будет оформляться в виде так называемых «методов», «функций» или «процедур», что соответствует обычному структурному программированию, однако теперь они могут (а во многих языках обязаны) принадлежать тому или иному классу. Например, по возможности, класс «строка текста» будет содержать все основные методы/функции/процедуры, предназначенные для работы со строкой текста, такие как поиск в строке, вырезание части строки и т.д.

Как и поля, код в виде методов/функций/процедур, принадлежащих классу, может быть отнесен либо к самому классу, либо к экземплярам класса. Метод, принадлежащий классу и соотнесенный с классом (статический метод) может быть вызван сам по себе и имеет доступ к статическим переменным класса. Метод, соотнесенный с экземпляром класса (обычный метод), может быть вызван только у самого объекта, и имеет доступ как к статическим полям класса, так и к обычным полям конкретного объекта (при вызове этот объект передастся скрытым параметром метода). Например, общее количество созданных строк можно узнать из любого места программы, но длину конкретной строки можно узнать только указав, тем или иным образом, длину какой строки будем мерить.

На практике объектно-ориентированное программирование сводится к созданию некоторого количества классов, включая интерфейс и реализацию, и последующему их использованию. Графическое представление некоторого количества классов и связей между ними называется диаграммой классов.

В ходе проектирования были реализованы следующие классы:

* Figure, который наследуется от базового класса TImage и является абстрактным классом;
* *Circle,* который наследуется от абстрактного класса Figure;

Классы представлены графически в виде диаграммы:

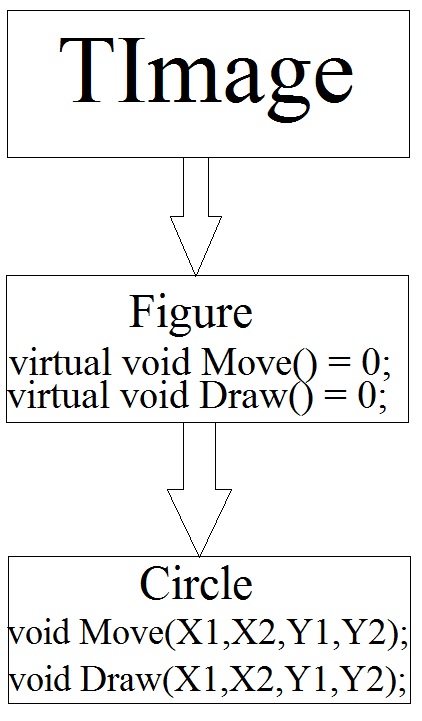


Рисунок 3 Диаграмма классов

## 2.2 Среда разработки

Платформой для реализации программного продукта стала *Borland C++ Builder 6*. Платформа является мощной и надежной средой быстрой разработки высокоэффективных приложений и включает обширный набор средств, которые повышают производительность труда программистов и сокращают продолжительность цикла разработки. Многофункциональная интегрированная среда разработки *C++Builder* включает компилятор, удовлетворяющий стандарта *ANSI/ISO*, встроенный дизайнер форм, обширный набор средств для работы с компонентами, инструмент *ObjectInspector*, менеджер проектов и отладчик.

**C++ Builder** — программный продукт, инструмент быстрой разработки приложений (RAD), интегрированная среда программирования (IDE), система, используемая программистами для разработки программного обеспечения на языке программирования C++.

Изначально разрабатывался компанией Borland Software, а затем её подразделением CodeGear, ныне принадлежащим компании Embarcadero Technologies.

C++ Builder объединяет в себе комплекс объектных библиотек (STL, VCL, CLX, MFC и др.), компилятор, отладчик, редактор кода и многие другие компоненты. Цикл разработки аналогичен Delphi. Большинство компонентов, разработанных в Delphi, можно использовать и в C++ Builder без модификации, но обратное утверждение не верно.

C++ Builder содержит инструменты, которые при помощи drag-and-drop действительно делают разработку визуальной, упрощает программирование благодаря встроенному WYSIWYG — редактору интерфейса и пр.

## 2.3 Реализация программного продукта

В данной курсовой работе был реализован абстрактный класс *Figure*, который наследуется от базового класса *TImage*.

В этом случае наследование позволяет использовать все методы, которые существуют у класса *TImage*, а в частности метод *Ellipse*(*Х1,Y1,X2,Y2)*, которым отрисовывается эллипс или круг. Параметры *Х1, Y1, X2, Y2* определяют координаты прямоугольника, внутри которого рисуется эллипс или, если прямоугольник является квадратом, − окружность.

На самом же деле круг является частным случаем эллипса и параметры *Х1, Y1, X2, Y2* определяют является ли этот эллипс кругом или нет.

Итоговая реализация программного продукта осуществлена классом *Circle*. Внутри этого класса реализовано два метода: *Move()* и *Draw(),* которые передвигают объект по экрану и, соответственно, отрисовывают его. Структура класса *Circle* отвечает за взаимодействие графической составляющей с внутренней частью реализации.

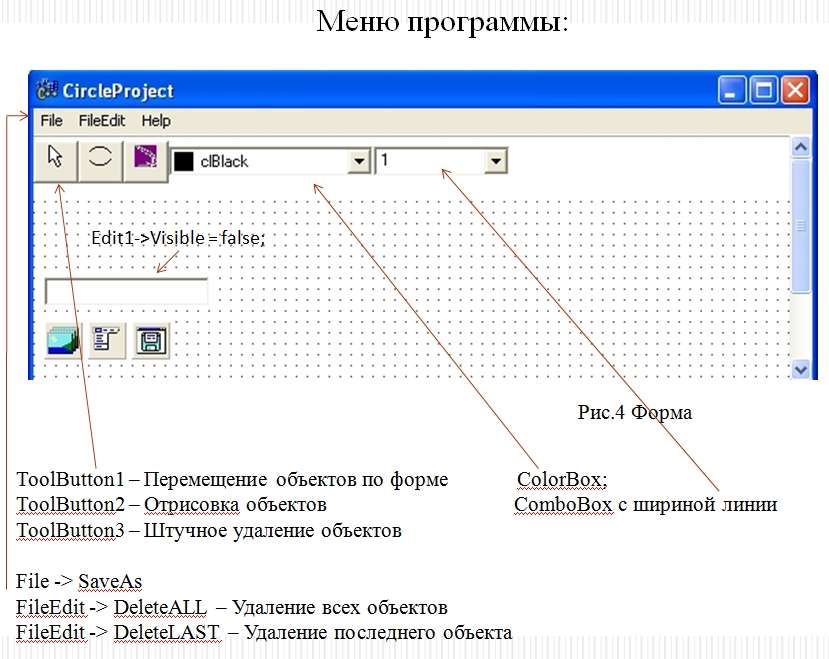
Контейнером для хранения объектов стал *vector,* который представляет собой динамический массив с определенным набором полезных базовых функций. Из данного контейнера реализовано удаление последнего элемента, удаление всех элементов и удаление отдельно выбранного (левым кликом мышки) элемента.

При выделении на *ToolBar’e* кнопки с изображением эллипса, можно рисовать эллипсы по нажатию левой кнопки мыши (ЛКМ) на область формы. При нажатии на ЛКМ в контейнер заносится динамически созданный объект типа *TImage*, который будет храниться там до тех пор, пока пользователь его не удалит либо не выйдет из программы.

Управление данным классом происходит посредством стандартного класса *C++Builder*– *TMainForm*.

## 2.4 Пользовательский интерфейс

Пользовательский интерфейс программы представлен небольшим меню, реализованным посредством класса *TMainForm*:



На форму был добавлен компонент *ToolBar,* на котором располагаются кнопки *ToolButton.* Им был присвоен режим *AllowUP*, который не позволяет нажимать несколько кнопок одновременно, что значительно упрощает написание кода для программиста.

Также добавлен компонент *MainMenu*, который позволяет добавлять \ редактировать главные кнопки программы ( например File->Exit и др.)

## 2.5 Тестирование программного продукта

Тестирование программного продукта проводилось как параллельно его разработке, так и отдельно. К отдельному консольному проекту подключались некоторые файлы, необходимые для тестирования работы вместе с различными контейнерами.

В процессе тестирования появлялись баги, среди которых основным из них был баг работы с оперативной памятью (ОЗУ). Следует сказать, что просто так удалять из контейнера какой-либо элемент нельзя, это должно сопровождаться переносом всех указателей на соответствующие элементы, поэтому при удалении конкретного объекта могут появляться ошибки работы с ОЗУ, но в целом программа работает и удаляет именно тот элемент, по которому кликнул пользователь. Естественно, что данная ошибка является значительным минусом программы, но исправить её так и не удалось.

# Заключение

В процессе написания курсовой работы на тему «Реализация объекта “Круг” на языке программирования C++» поставленная цель была достигнута.

При достижении поставленной цели были решены следующие задачи:

* изучены теоретические сведения, необходимые для достижения заданной цели;
* систематизированы и обобщены полученные знания;
* создан программный продукт, реализующий объект «Круг»;
* проведены тестирование и отладка программы;

В ходе выполнения курсовой работы были изучены особенности реализации различных объектов, а в ходе тестирования созданного программного продукта были выявлены следующие недостатки:

1. Уведомления, связанные с ошибкой работы с памятью при удалении конкретного объекта.
2. Не реализовано выделение мышью нескольких объектов.

Также следует отметить, что в процессе написания курсовой работы были изучены следующие темы, не относящиеся к задачам, но нашедшие в ней довольно широкое применение:

* основы работы с *STL*-контейнерами (*vector*, *set*, *deque*, *map*, *list*);
* генерация и обработка исключений.

Подводя итоги, можно сказать, что получившийся программный продукт реализует ключевые аспекты моделируемого объекта «Круг». Продукт довольно прост в освоении, не требующий каких-либо специфичных знаний.

# Список литературы

1. Cyberforum [Электронный ресурс]. – Форум программистов и сисадминов. – Режим доступа: <http://www.cyberforum.ru>
2. Википедия [Электронный ресурс]. – Свободная энциклопедия. – Режим доступа: <https://ru.wikipedia.org/wiki/>
3. В.А. Скляров. Язык C++ и объектно–ориентированное программирование: Справочное издание. – Минск: Высшая школа, 1997.
4. Т. Фейсон. Объектно-ориентированное программирование на Borland C++ 4.5: Пер. с англ. - Киев: Диалектика, 1996.
5. RationalRose[Электронный ресурс]. – Диаграммы классов. – Режим доступа: <http://itteach.ru/rational-rose/diagrammi-klassov>
6. Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы. Построение и анализ. 2-е издание. Пер. с англ. 2005г. —863с.
7. Архангельский А. Я. Язык C++ в C++ Builder : справочное и методическое пособие / А. Я. Архангельский. - М., 2008. - 942 с. : ил.
8. Архангельский А. Я. Программирование в C++ Builder 6 и 2006 / А. Я. Архангельский, М. А. Тагин. - М., 2007. - 1181 с.
9. Седжвик Р. Фундаментальные алгоритмы на C++. Ч. 5 : [пер. с англ.] / Роберт Седжвик. - М. [и др.], 2002. - 484 с. : ил.

ЛИСТИНГ

**Figure.h**

#include <vcl.h>

#ifndef FigureH

#define FigureH

class Figure : public TImage

{

virtual void Draw() = 0;

virtual void Move() = 0;

};-

#endif

**Circle.h**

#ifndef CircleH

#define CircleH

#include <vcl.h>

#include "Figure.h"

class Circle : public Figure

{

public :

int X1, Y1, X2, Y2;

void Move(TObject \*Sender, int X, int Y, int X1, int Y1, TShiftState Shift);

void Draw(TImage\* img, int X1, int Y1, int X2, int Y2, TColor Color, int Size);

};#endif

**Circle.cpp**

#include "Circle.h"

#pragma hdrstop

void Circle::Move(TObject \*Sender, int X, int Y, int X1, int Y1, TShiftState Shift)

{

TImage \*moveImage = ((TImage\*)Sender);

if (Shift.Contains(ssLeft))

{

moveImage->Left += X - X1;

moveImage->Top += Y - Y1;

}

}

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

void Circle::Draw(TImage\* img, int X1, int Y1, int X2, int Y2, TColor Color, int Size)

{

if (X2 - X1 > 0)

img->Left = X1;

else

img->Left = X2;

if (Y2 - Y1 > 0)

img->Top = Y1;

else

img->Top = Y2;

img->Width = abs(X2 - X1);

img->Height = abs(Y2 - Y1);

img->Transparent = true;

img->Canvas->Pen->Color = Color;

img->Canvas->Pen->Width = Size;

img->Canvas->Ellipse(0, 0, abs(X2 - X1), abs(Y2 - Y1));

}

#pragma package(smart\_init)

**MainForm.h**

#ifndef MainFormH

#define MainFormH

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

#include <Classes.hpp>

#include <Controls.hpp>

#include <StdCtrls.hpp>

#include <Forms.hpp>

using namespace std;

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

class TForm1 : public TForm

{

\_\_published: // IDE-managed Components

TEdit \*Edit1;

TImageList \*ImageList1;

TToolBar \*ToolBar1;

TToolButton \*ToolButton1;

TToolButton \*ToolButton2;

TColorBox \*ColorBox1;

TComboBox \*ComboBox1;

TSaveDialog \*SaveDialog1;

TMainMenu \*MainMenu1;

TMenuItem \*File1;

TMenuItem \*FileEdit1;

TMenuItem \*Help1;

TMenuItem \*SaveAs1;

TMenuItem \*NOHELP1;

TMemo \*Memo1;

TMemo \*Memo2;

TToolButton \*ToolButton3;

TMenuItem \*DeleteALL1;

TMenuItem \*DeleteLAST1;

void \_\_fastcall ToolButton1Click(TObject \*Sender);

void \_\_fastcall ToolButton2Click(TObject \*Sender);

void \_\_fastcall ToolButton3Click(TObject \*Sender);

void \_\_fastcall DeleteALL1Click(TObject \*Sender);

void \_\_fastcall FormClose(TObject \*Sender, TCloseAction &Action);

void \_\_fastcall Image1Click(TObject \*Sender);

void \_\_fastcall FormMouseDown(TObject \*Sender, TMouseButton Button,

TShiftState Shift, int X, int Y);

void \_\_fastcall FormMouseMove(TObject \*Sender, TShiftState Shift,

int X, int Y);

void \_\_fastcall FormMouseUp(TObject \*Sender, TMouseButton Button,

TShiftState Shift, int X, int Y);

void \_\_fastcall DeleteLAST1Click(TObject \*Sender);

private:

public:

int rectX1, rectY1, rectX2, rectY2;

int X1, Y1, X2, Y2;

int Moving, CirclePainting, Selecting;

Circle \*Circle1;

TImage\* img;

TImage\* temp;

vector <TImage\*> :: iterator iterator;

vector <TImage\*> AllCircles;

vector <TImage\*> SelectedCircles;

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

};

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

extern PACKAGE TForm1 \*Form1;

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

#endif

**MainForm.cpp**

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

#include <vcl.h>

#pragma hdrstop

#include "MainForm.h"

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

#pragma package(smart\_init)

#pragma resource "\*.dfm"

TForm1 \*Form1;

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

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

: TForm(Owner)

{

Form1->DoubleBuffered = true;

TImage\* img = NULL;

Moving = false; CirclePainting = false; Selecting = false;

}

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

void \_\_fastcall TForm1::ToolButton1Click(TObject \*Sender)

{

Moving = true;

CirclePainting = false;

Selecting = false;

}

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

void \_\_fastcall TForm1::ToolButton2Click(TObject \*Sender)

{

Moving = false;

CirclePainting = true;

Selecting = false;

}

void \_\_fastcall TForm1::ToolButton3Click(TObject \*Sender)

{

Moving = false;

CirclePainting = false;

Selecting = true;

}

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

void \_\_fastcall TForm1::FormMouseDown(TObject \*Sender, TMouseButton Button,

TShiftState Shift, int X, int Y)

{

X1 = X; rectX1 = X;

Y1 = Y; rectY1 = X;

}

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

void \_\_fastcall TForm1::FormMouseMove(TObject \*Sender, TShiftState Shift,

int X, int Y)

{

if (Moving == true)

Circle1->Move(Sender, X, Y, X1, Y1, Shift);

}

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

void \_\_fastcall TForm1::FormMouseUp(TObject \*Sender, TMouseButton Button,

TShiftState Shift, int X, int Y)

{

X2 = X; rectX2 = X;

Y2 = Y; rectY2 = X;

if (CirclePainting == true)

{

img = new TImage(this);

int Size = ComboBox1->ItemIndex;

Circle1->Draw(img, X1, Y1, X2, Y2, ColorBox1->Selected, Size);

AllCircles.push\_back(img);

img->Parent = Form1;

img->OnMouseDown = FormMouseDown;

img->OnMouseMove = FormMouseMove;

img->OnClick = Image1Click;

Memo1->Lines->Add(IntToStr(AllCircles.size()));

}

}

void \_\_fastcall TForm1::Image1Click(TObject \*Sender)

{

if (Selecting == true)

{

if (img = dynamic\_cast <TImage\*> (Sender))

{

SelectedCircles.push\_back(img);

for (unsigned int i = 0; i < AllCircles.size(); i++)

{

for (unsigned int j = 0; j < SelectedCircles.size(); j++)

{

if (AllCircles[i] == SelectedCircles[j]) // pointer values compare

{

delete AllCircles[i];

AllCircles.erase(AllCircles.begin() + i);

}

}

} Memo2->Lines->Add(IntToStr(SelectedCircles.size()));

}

}

}

void \_\_fastcall TForm1::DeleteALL1Click(TObject \*Sender)

{

for(unsigned int i = 0; i < AllCircles.size(); i++)

delete AllCircles[i];

AllCircles.clear();

}

void \_\_fastcall TForm1::FormClose(TObject \*Sender, TCloseAction &Action)

{

for(unsigned int i = 0; i < AllCircles.size(); i++)

delete AllCircles[i];

AllCircles.clear();

}

void \_\_fastcall TForm1::DeleteLAST1Click(TObject \*Sender)

{

if(AllCircles.empty() == false)

{

delete this->img;

AllCircles.pop\_back();

this->img = AllCircles.back();}}