**КУРСОВАЯ РАБОТА**

**по дисциплине “Объектно-ориентированное программирование”**

**ОГЛАВЛЕНИЕ**

[**1. Поста**](#_Toc515570163)**новка задачи (вариант №8) 1**

**a. Описание задания**  **a**

**b. Вариант выполнения задания**  **b**

**c. Описание иерархии объектов и методов объектов**  **с**

**2**[**.**](#_Toc515570163) **Описание алгоритма программы**  **2**

**Подпункт 1**  **d**

**Подпункт 2**  **e**

**3**[**.**](#_Toc515570163) **Скриншоты работы программы**  **3**

**Подпункт 1**  **g**

**Подпункт 2**  **h**

**Подпункт 3**  **o**

**4**[**.**](#_Toc515570163) **Листинг кода**  **3**

**Подпункт 1**  **g**

**Подпункт 2**  **h**

**Подпункт 3**  **o**

**Подпункт 4**  **g**

**Подпункт 5**  **h**

**Подпункт 6**  **o**

**Подпункт 7**  **g**

**Подпункт 8**  **h**

**Подпункт 9**  **o**

[**1. Поста**](#_Toc515570163)**новка задачи**

***a.*** *Описание задания*.

Круговое движение с вращением составного графического объекта.

***b.*** *Вариант выполнения задания*.

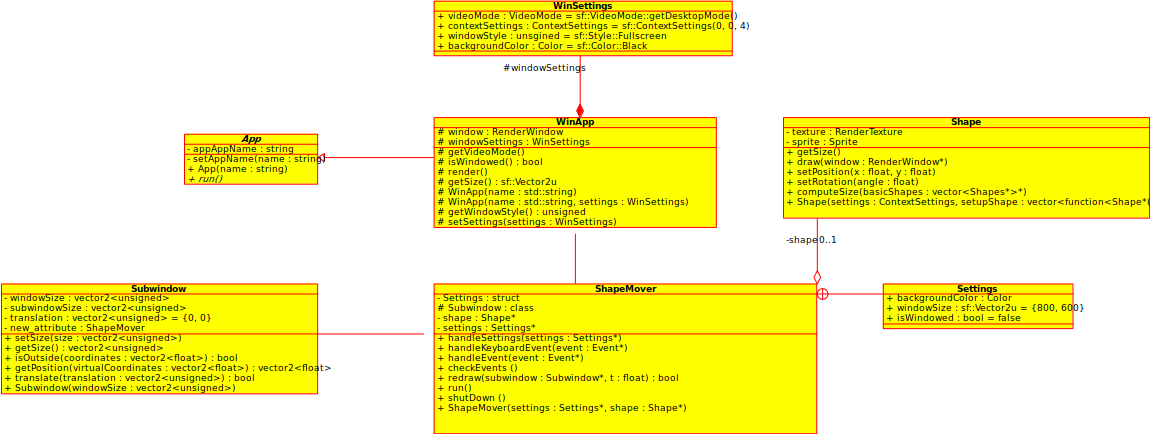
Обозначим объект, который будет двигаться и вращаться. Для реализации кругового движения (движения по окружности) используем математические (физические) формулы движения точки по окружности *x = x0 + rcos(t)*, *y = y0 + rsin(t)*, где x0, y0 – центральная точка окружности. Используем для реализации программы библиотеку SFML (<https://en.sfml-dev.org/>), которая предоставляет пользователю удобный интерфейс для работы с окном приложения и простейшими геометрическими фигурами.

Т. к. математические формулы движения по окружности подразумевают точку, а не объект с заданным размером, то сведем задачу к работе с точкой. Но для начала определим размер и форму фигуры. Фигура составная, следовательно, строится путем комбинации простейших геометрических фигур. В этом варианте задания я реализую нечто похожее на космический корабль, как показано в примере. И оно будет двигаться и вращаться. Для двигателей используем прямогульники, для “”тела” корабля – круг, для “носа” – треугольник. Для удобства при проверке выхода за границы видимой области обозначим форму фигуры как прямоугольник наименьшей площади, такой, что в него помещается вся наша составная фигура. Обозначим точку, которая находится на пересечении левой и верхней сторон нашего прямогульника. Она и будет двигаться по окружности.

Для того, чтобы наша фигура не уходила за границы окна, задаим рабочую область. Т. к. мы работаем с точкой на левом верхнем углу фигуры, импровизированного прямоугольника, то ограничичим левую и нижнюю части окна в использовании. Вычтем из реальных высоты и длины окна высоту и длину заданной фигуры, и получим размер области, с которой будем работать. Кстати, можно урезать рабочую область видимости до квадрата, у которого стороны будут равны длине наименьшей стороны ещё неурезанной рабочей области. Квадрат будет наименьшей площади, такой, что в него помещается вся окружность. И для более приятного глазу вида можно центрировать урезанную рабочую область относительно неурезанной. Таким образом, картинка будет находится по центру, а сама окружность будет смещена чуть левее и выше, как раз на размер фигуры. Для реализации работы с урезанной рабочей областью, меньшей, чем размер окна, создадим отдельный класс.

***c.*** *Описание иерархии объектов и методов объектов.*

Иерархия классов, объектов и их методов следующая (на изображении представлена uml диаграмма, а также [в файле](../../umbrello/Shape%20mover.svg) “Shape mover.svg”):



Ред. 1. Я добавил классы настроек для класса окна и класса приложения. Настройки проложения передаются в конструктор по ссылке и приложение хранит ссылку. Методы объекта окна используют конструктор копирования и, в случае, если приложение не задаёт настройки окна, обект настроек в любом случае создаётся при создании объекта окна, т.е. обект окна имеет заданные настройки в любом случае. Это сделано для того, чтобы перегрузить конструктор класса окна и дать возомжность пользователю не создавать настройки вручную. А класс приложения имеет единственный конструктор, который обязательным параметром имеет ссылку на объект настроек.

Таким образом, мы имеем класс приложения, класс фигуры и промежуточные классы. Для работы приложения помимо класса фигуры используются вспомогательный класс виртуального окна и базовый класс оконного приложения, который, в свою очередь, имеет базовый абстрактный класс приложения, который должен быть реализован, как например, класс оконного приложения. Но процесс работы класс оконного приложения не описывает, а делегирует данную задачу классу-потомку, вследствие чего и сам является абстрактным классом. Класс, описывающий процесс работы программы, использует все вохможности как базового класса, так и класса фигуры, и, по сути, является основой программы, где и задействуются все пользовательские данные, как, например, описание фигуры, размер окна и т. д.

**2**[**.**](#_Toc515570163) **Описание алгоритма программы**

***a.*** *Алгоритм программы*

Учитывая иерархию классов, объектов и их методов, алгоритм работы программы следующий:

***b.*** *Выводы по разделу #2*

Так как каждый описанный выше этап работы программы имеет вполне определённое поведение при различных входных данных (за исключением багов, недочётов и уязвимостей которые будут выявляться в ходе эксплуатации программы при условии поддержки и развития проиложенния разработчиком, то есть мной (-: ), можем считать процесс работы данной программы алгоритмом.

**3**[**.**](#_Toc515570163) **Скриншоты работы программы**

**4**[**.**](#_Toc515570163) **Листинг кода**