Московский авиационный институт (национальный исследовательский университет)

Факультет прикладной математики и физики

Кафедра вычислительной математики и программирования

Лабораторная работа №1 по курсу «Объектно-ориентированное программирование»

Студентка: А. Довженко
Преподаватель: А. В. Поповкин
Группа: 08-207

Вариант: 12
Дата:
Оценка:
Подпись:

Москва, 2017

Лабораторная работа №1

Задача: Необходимо спроектировать и запрограммировать на языке C++ классы фигур, согласно варианту задания.
Классы должны удовлетворять следующим правилам:

- Должны иметь общий родительский класс Figure.
- Должны иметь общий виртуальный метод Print, печатающий параметры фигуры и ее тип в стандартный поток вывода cout.
- Должны иметь общий виртуальный метод расчета площади фигуры — Square.
- Должны иметь конструктор, считывающий значения основных параметров фигуры из стандартного потока cin.
- Должны быть расположены в раздельных файлах: отдельно заголовки (.h), отдельно описание методов (.cpp).

Фигуры: трапеция, ромб, пятиугольник.

1 Описание

Основная идея ООП — **объект**. Объект есть сущность, одновременно содержащая данные и поведение. Они являются строительными блоками объектно-ориентированных программ. Та или иная программа, которая задействует объектно-ориентированную технологию, по сути является набором объектов. Перед тем как создать объект C++, необходимо определить его общую форму, используя ключевое слово class. Класс определяет новый пользовательский тип данных, который соединяет в себе код и данные.

Классы в программировании состоят из свойств (атрибутов) и методов. **Свойства** — это любые данные, которыми можно характеризовать объект класса. **Методы** — это функции, которые могут выполнять какие-либо действия над данными (свойствами) класса. Компоненты, использованные при определении класса, называются его членами. Как правило, класс имеет интерфейс и реализацию. Интерфейс — это часть объявления класса, к которой его пользователь имеет прямой доступ. Реализация — это часть объявления класса, доступ к которой пользователь может получить только косвенно, через интерфейс. Открытый интерфейс индентифицируется меткой **public:**, а реализация меткой **private:**.

В ООП существует 3 основных принципа построение классов:

- **Инкапсуляция** — это свойство, позволяющее объединить в классе и данные, и методы, работающие с ним и скрыть детали реализации от пользователя.
- **Наследование** — это свойство, позволяющее создать новый класс-потомок на основе уже существующего, при этом все характеристики класса-родителя присваиваются классу-потомку.
- **Полиморфизм** — свойство классов, позволяющее использовать объекты классов с одинаковым интерфейсом без информации о типе и внутреннй структуре объекта.

Проблема поддержания правильного состояния переменных актуальна и для самого первого момента выставления начальных значений. Для этого в класса предусмотрены специальные методы/функции, называемые конструкторами. **Конструктор** является специальной функцией-членом класса, которая должна определяться с тем же именем, что и класс, чтобы компилятор мог отличить его от других функций класса. Важным отличием конструкторов от других функций является то, что конструкторы не возвращают значений, так что они не могут специфицировать возвращаемый тип. C++ требует вызова конструктора для каждого создаваемого объекта, что позволяет гарантировать корректную инициализацию объекта перед тем, как он будет использоваться в программе — когда объект создается, вызов конструктора происходит автоматически. Любому классу, который не определяет конструктор явным

образом, компилятор предоставляет конструктор по умолчанию, т.е. конструктор без параметров. **Деструктор** класса вызывется при уничтожении объекта. Имя деструктора аналогично имени конструктора, только в начале ставится знак тильды. Деструктор не имеет входных параметров.

Виртуальная функция — это функция-член, которая, как предполагается, будет переопределена в производных классах. При обращении к объекту производного класса, используя указатель или ссылку на базовый класс, можно вызвать виртуальные функции объекта и выполнить переопределенную в производном классе версию функции. Виртуальные функции гарантируют, что выбрана верная версия функции для объекта, независимо от выражения, используемого для вызова функции.

Операции ввода/вывода выполняются с помощью классов istream (потоковые ввод) и ostream (потоковый вывод). Третий класс, iostream, является производным от них и поддерживает двунаправленный ввод/вывод. Для удобства в библиотеке определены три стандартных объекта-потока: cin — объект класса istream, соответствующий стандартному вводу; cout — объект класса ostream, соответствующий стандартному выводу; cerr — объект класса ostream, соответствующий стандартному выводу для ошибок. Вывод осуществляется, как правило, с помощью перегруженного оператора сдвига влево, а ввод — с помощью оператора сдвига вправо.

2 Исходный код

trapeze.cpp	
Trapeze();	Конструктор класса
Trapeze(std::istream &is);	Конструктор класса из стандартного потока
Trapeze(const Trapeze& orig);	Конструктор копии класса
double Square();	Площадь фигуры
void Print();	Печать фигуры
~Trapeze();	Деконструктор класса
rhomb.cpp	
Rhomb();	Конструктор класса
Rhomb(std::istream &is);	Конструктор класса из стандартного потока
Rhomb(const Rhomb& orig);	Конструктор копии класса
double Square();	Площадь фигуры
void Print();	Печать фигуры
~Rhomb();	Деконструктор класса
pentagon.cpp	
Pentagon();	Конструктор класса
Pentagon(std::istream &is);	Конструктор класса из стандартного потока
Pentagon(const Pentagon& orig);	Конструктор копии класса
double Square();	Площадь фигуры
void Print();	Печать фигуры
~Pentagon();	Деконструктор класса

```
1
2  class Figure
3  {
4  public:
5      virtual double Square() = 0;
6      virtual void Print() = 0;
7      virtual ~Figure() {};
8  };
9
10 class Trapeze : public Figure
11 {
12 public:
13     Trapeze();
14     Trapeze(std::istream &is);
15     Trapeze(int32_t small_base, int32_t big_base, int32_t l_side, int32_t r_side);
16     Trapeze(const Trapeze& orig);
```

```
17      double Square() override;
18      void Print() override;
19      virtual ~Trapeze();
20
21  private:
22      int32_t small_base;
23      int32_t big_base;
24      int32_t l_side;
25      int32_t r_side;
26  };
27
28
29  class Rhomb : public Figure
30  {
31  public:
32      Rhomb();
33      Rhomb(std::istream &is);
34      Rhomb(int32_t side, int32_t smaller_angle);
35      Rhomb(const Rhomb& orig);
36      double Square() override;
37      void Print() override;
38      virtual ~Rhomb();
39
40  private:
41      int32_t side;
42      int32_t smaller_angle;
43  };
44
45  class Pentagon : public Figure
46  {
47  public:
48      Pentagon();
49      Pentagon(std::istream& is);
50      Pentagon(int32_t side);
51      Pentagon(const Pentagon& orig);
52      double Square() override;
53      void Print() override;
54      virtual ~Pentagon();
55
56  private:
57      int32_t side;
58  };
```

3 Консоль

```
karma@karma:~/mai_study/OOP/lab1$ valgrind --leak-check=full ./run
==5388== Memcheck,a memory error detector
==5388== Copyright (C) 2002-2015,and GNU GPL'd,by Julian Seward et al.
==5388== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==5388== Command: ./run
==5388==
Use 'help'or 'h'to get help.
h
Commands 'create_trapeze'and 'cr_tr'create new trapeze with your parameters.
Commands 'create_rhomb'and 'cr_rh'create new rhomb with your parameters.
Commands 'create_pentagon'and 'cr_pen'create new pentagon with your parameters.
Commands 'print_trapeze'and 'pr_tr'output parameters of trapeze.
Commands 'print_rhomb'and 'pr_rh'output parameters of rhomb.
Commands 'print_pentagon'and 'pr_pen'output parameters of pentagon.
Commands 'square_trapeze'and 'sq_tr'output square of trapeze.
Commands 'square_rhomb'and 'sq_rh'output square of rhomb.
Commands 'square_pentagon'and 'sq_pen'output square of pentagon.
Commands 'quit'and 'q'exit the program.
cr_tr
Enter smaller base,bigger base,left side and right side.
2 7 4 4
pr_tr
Smaller base = 2,bigger base = 7,left side = 4,right side = 4
cr_pen
Enter side.
10000
sq_pen
Square: 1.72048e+08
cr_pen
Enter side.
9
pr_pen
Sides =  9
cr_rh
Enter side and smaller angle.
15 30
pr_rh
Side = 15,smaller_angle = 30
pr_tr
```

```
Smaller base = 2,bigger base = 7,left side = 4,right side = 4
pr_pen
Sides =  9
sq_tr
Square: 16
sq_rh
Square: 112.5
sq_pen
Square: 139.359
q
Trapeze deleted
Rhomb deleted
Pentagon deleted
==5388==
==5388== HEAP SUMMARY:
==5388==     in use at exit: 72,704 bytes in 1 blocks
==5388==   total heap usage: 7 allocs,6 frees,74,824 bytes allocated
==5388==
==5388== LEAK SUMMARY:
==5388==    definitely lost: 0 bytes in 0 blocks
==5388==    indirectly lost: 0 bytes in 0 blocks
==5388==      possibly lost: 0 bytes in 0 blocks
==5388==    still reachable: 72,704 bytes in 1 blocks
==5388==         suppressed: 0 bytes in 0 blocks
==5388== Reachable blocks (those to which a pointer was found) are not shown.
==5388== To see them,rerun with: --leak-check=full --show-leak-kinds=all
==5388==
==5388== For counts of detected and suppressed errors,rerun with: -v
==5388== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

4 Выводы

В данной работе я познакомилась с новой для себя, объектно-ориентированной парадигмой программирования. В отличие от знакомой мне процедурной парадигмы, где атрибуты и поведения обычно разделяются, при объектно-ориентированном проектировании атрибуты и поведения размещаются в рамках одного объекта. Конечно, они не являются взаимоисключающими, и многие программы, написанные на C++, принадлежат обеим парадигмам. Я приобрела навыки проектирования классов и работы с ними. Фундаментальные концепции ООП —инкапсуляция, наследование и полиморфизм — также отражены в моей работе. Инкапсуляция в виде разделения интерфейса (печать параметров фигуры и подсчет площади) и реализации (параметры фигуры), наследование в виде производных классов Trapeze, Rhomb и Pentagon от класса Figure, полиморфизм в виде переопределении методов Print и Square. Удобно использовать объекты, не задумываясь о внутренней реализации.