# ОЦЕНКА ХАРАКТЕРИСТИК ПРОГРАММ НА ОСНОВЕ ОБЪЕКТНО- ОРИЕНТИРОВАННЫХ МЕТРИК

## МЕТРИКИ ЧИДАМБЕРА И КЕМЕРЕРА

### Теоретические сведения

В 1994 г. С. Чидамбер и К. Кемерер (Shyam R. Chidamber и Cris F. Кетегег) предложили шесть проектных метрик, основанных на анализе методов класса, дерева наследования и других характеристик объектно-ориентированной среды.

*Взвешенные методы на класс WMC* (Weighted Methods Per Class). Данная метрика позволяет измерять сложность классов с учетом сложности их методов. Метрика называется взвешенной, потому что весом метода считается количественная характеристика сложности метода.

Пусть в классе С определены $\boldsymbol{n}$ методов со сложностью $c[1], c[2], \dots, c[n]$. Для оценки сложности может быть выбрана любая метрика сложности (например, Холстеда или цикломатическая сложность в зависимости от интересующего критерия). Основным при этом является процесс нормализации этой метрики таким образом, чтобы номинальная сложность для метода принимала значение 1. В этом случае количество методов и их сложность могут служить своеобразным индикатором затрат, необходимых на реализацию и тестирование классов.

Кроме того, с ростом числа методов возрастает сложность дерева наследования, поскольку все подклассы должны наследовать методы их родителей. С ростом количества методов в классе применение этого класса становится все более специфичным, ограничивается возможность его многократного использования. По этим причинам требование к этой метрике таково: *метрика* **wmc** *должна иметь разумно низкое значение*.

В простейшем случае сложность метода можно определить количеством строк текста этого метода в анализируемой программе. Пусть $\boldsymbol{L}[\boldsymbol{i}]$ характеризует количество строк текста у метода с номером $i$, а всего в классе содержится $\boldsymbol{n}$ методов. Тогда для этой метрики

измерения методов сложность класса K вычисляется следующим образом:

$$WMC(K) = L[0] + L[1] + \cdots + L[n].$$

Нетрудно заметить, что для измерения сложности класса метрика *wmc* зависит от способа измерения сложности методов этого класса.

Часто используется разновидность метрики *wmc*, когда все методы имеют одинаковый вес. В этом случае метрика носит наименование «Количество методов на класс» ***NM*** (Number of Methods). Метрика ***NM*** применяется для измерения сложности классов на ранних этапах разработки системы, когда еще нет детальной информации о применяемых методах. В качестве практических рекомендаций специалисты советуют на этапе проектирования системы разделять классы со слишком большим количеством методов на несколько классов для снижения сложности используемых методов.

Оказывается, что подсчитывать количество методов в классе достаточно сложно. Возможны два противоположных варианта учета.

1. Подсчитываются только методы текущего класса. Унаследованные методы игнорируются, поскольку они считаются уже подсчитанными в тех классах, где определялись. В связи с этим лучшим показателем функциональных возможностей класса становится инкрементность класса, т. е. возможность его пошагового развития. Для понимания того, что делает класс, основным источником информации являются его собственные операции. Если класс не может отреагировать на сообщение самостоятельно (например, в нем отсутствует собственный метод), этот класс направляет сообщение своему родителю.
2. Подсчитываются как методы, определенные в текущем классе, так и все унаследованные методы. Этот подход подчеркивает важность пространства состояний в понимании класса (а не инкрементности класса).

На практике приемлем любой из описанных вариантов. Единственным условием является постоянство применения того или иного метода, чтобы не приходилось менять вариант учета при переходе к новому проекту. В этом случае будет обеспечен корректный сбор метрических данных.

Глубина дерева наследования ***DIT*** (Depth of Inheritance Tree) позволяет определить количество классов-предков, которые

потенциально оказывают влияние на данный класс. Другими словами, эта метрика характеризует самый длинный путь по иерархии классов к данному классу от класса-предка. Этот показатель должен быть возможно большим, так как при большей глубине возрастает абстракция данных, снижается насыщенность класса методами. В то же время не следует забывать, что при достаточно большой глубине существенно возрастает сложность понимания и написания программы.

Количество потомков $\boldsymbol{NOC}$ (Number Of Child) позволяет определить количество непосредственных потомков данного класса.

С ростом значения $\boldsymbol{NOC}$ возрастает многократность использования, поскольку наследование представляет собой одну из форм повторного использования. В то же время при возрастании метрики $\boldsymbol{NOC}$ снижается уровень абстракции родительского класса. Это означает, что в действительности некоторые из детей уже могут не быть элементами родительского класса, вследствие чего возможно их некорректное использование. Кроме того, количество детей характеризует потенциальное влияние класса на проект. По мере роста $\boldsymbol{NOC}$ возрастает количество тестов, необходимых для проверки каждого ребенка.

Метрики $\boldsymbol{DIT}$ и $\boldsymbol{NOC}$ являются количественными характеристиками формы и размера структуры классов. *Хорошо структурированная объектно-ориентированная система чаще организуется в виде леса классов, нежели как очень высокое дерево*.

Связанность между классами объектов $\boldsymbol{CBO}$ (Coupling Between Object classes) дает возможность определить количество классов, с которыми связан данный класс. Это имеет существенное значение, когда один класс использует методы или экземпляры другого класса. Для данной метрики справедливы все утверждения, введенные ранее при характеристике связанности модулей. Другое определение метрики имеет следующую трактовку: $\boldsymbol{CBO}$ равно количеству сцеплений класса, а сцепление организует вызов метода или свойства в другом классе.

Данная метрика характеризует статическую составляющую внешних связей классов. С ростом значения $\boldsymbol{CBO}$ уменьшается абстракция данных, и многократное использование класса уменьшается.

Вполне очевидно, что чем более независим класс, тем легче его повторно использовать в другом приложении. Высокое значение

данной метрики усложняет модификацию и тестирование, которое неминуемо следует за модификацией программного продукта. Чем больше сцеплений, тем выше чувствительность всего проекта к изменениям в отдельных его частях. Минимизация сцеплений между объектами улучшает модульность и содействует инкапсуляции проекта.

$\boldsymbol{CBO}$ для каждого класса должно иметь разумно низкое значение. Это также согласуется с рекомендациями по уменьшению сцепления стандартного ПО.

Количество откликов на класс $\boldsymbol{RFC}$ (Response For Class) позволяет определить количество методов, которое может быть выполнено в ответ на получение сообщения данным классом. В этой метрике учитываются не только выполняемые методы данного класса, но и методы других классов, т. е. с помощью этой метрики можно оценить и степень потенциального «общения» конкретного класса с другими классами:

$$RFC = \{R\} + \{M\}$$

где $\{R\}$ - множество методов, вызываемых методом $r$ ; {M} - множество всех методов в классе.

Возможно другое толкование метрики: $RFC$ представляет собой количество методов класса, увеличенное на число методов других классов, вызываемых из данного класса.

Метрика $RFC$ является мерой потенциального взаимодействия данного класса с другими классами и позволяет судить о динамике поведения объекта в системе, т. е. данная метрика характеризует динамическую составляющую внешних связей классов. Если в ответ на сообщение может быть вызвано большое количество методов, то усложняются тестирование и отладка класса, так как от разработчика тестов в этом случае потребуется высокий уровень понимания класса. Кроме того, возрастает длина тестовой последовательности.

С ростом $RFC$ увеличивается сложность класса. Самая плохая величина отклика может использоваться при определении времени тестирования.

Отсутствие сцепления в методах $\boldsymbol{LCOM}$ (Lack Cohesion Of Methods) позволяет оценить зависимость методов класса друг от друга. Для вычисления этой метрики подсчитывается количество пар методов, которые не используют общие атрибуты класса. Затем подсчитывается количество пар методов, которые используют общие переменные. Метрика $\boldsymbol{LCOM}$ равна разности между первым числом и

вторым. Если при этом получается отрицательное число, то значение метрики считается равным нулю.

Каждый метод внутри своего класса обращается к одному или нескольким свойствам (экземплярным переменным). Метрика $\boldsymbol{LCOM}$ показывает, насколько методы не связаны друг с другом через свойства (переменные). Если все методы обращаются к одинаковым свойствам, то значение $\boldsymbol{LCOM = 0}$.

Связность методов внутри класса должна быть высокой, так как это содействует инкапсуляции. Если метрика $\boldsymbol{LCOM}$ имеет высокое значение, то методы слабо связаны друг с другом через свойства. Это увеличивает сложность, что ведет к возрастанию появления ошибок при проектировании. Высокие значения $\boldsymbol{LCOM}$ означают, что класс, очевидно, спроектирован не самым лучшим образом, и требуется разбить его на два или более отдельных класса. Вычисление $\boldsymbol{LCOM}$ помогает определить недостатки в проектировании классов, поскольку эта метрика характеризует качество упаковки данных и методов в оболочку класса. Следовательно, желательно сохранять высокий уровень связности в классе, добиваясь низкого значения метрики $\boldsymbol{LCOM}$.

# ПРИМЕРЫ

## Задача «Платеж за электроэнергию»

Необходимо определить класс, описывающий платеж за электроэнергию. В рамках класса следует предусмотреть следующие поля:

- фамилия плательщика;
- потребление электроэнергии за оплачиваемый месяц;
- нормативное среднемесячное потребление;
- тариф (стоимость одного киловатт-часа).

Рекомендуется применять следующие методы:

- вычисление суммы оплаты;
- формирование сводной информации по одному платежу (вид платежа, фамилия, сумма, потребление).

Платеж может выполняться по показаниям счетчика или по нормативно установленному среднемесячному потреблению. В процессе эксплуатации программы тариф и нормативно установленное среднемесячное потребление не изменяются. Для

создания конкретного платежа предусмотреть соответствующий конструктор.

Все платежи должны сохраняться в архиве. Запросы по ведению архива выполняются статическими методами класса «Запрос»:

- занесение платежей в архив. Данные платежа вводятся с клавиатуры. Ввод отрицательного показания счетчика означает оплату по нормативно установленному среднемесячному потреблению;
- вывод сводной информации из архива.

Архив моделируется массивом объектов.

В основном классе «Платежи» следует сформировать архив платежей. По данным архива предусмотреть выдачу сводной информации о платежах.

При решении задачи необходимо разработать исходный код программы, а также определить оценки характеристик программы на основе объектно-ориентированных метрик Чидамбера и Кемерера.

## Реализация программы

Текст программы на языке C# для реализации возможного алгоритма решения поставленной задачи представлен на рис. 1.

| Номера строк | Строки программы |
|---|---|
| 1 | using System; |
| 2 | using System.Collections.Generic; |
| 3 | using System.Text; |
| 4 | namespace EX1 |
| 5 | { |
| 6 | class Электро |
| 7 | { |
| 8 | private static double тариф = 1.84; |
| 9 | private static int потреблениеСреднее = 300; |
| 10 | private string фамилия; |
| 11 | private int потребление; |
| 12 | public Электро(string фамилия) |
| 13 | { |
| 14 | this.фамилия = фамилия; |
| 15 | потребление = потреблениеСреднее; |
| 16 | } |
| 17 | public Электро(string фамилия, int текущее, int предыдущее) |
| 18 | { |
| 19 | this.фамилия = фамилия; |
| 20 | потребление = текущее - предыдущее; |
| 21 | } |
| 22 | public double Сумма() |
| 23 | { |
| 24 | return потребление * тариф; |
| 25 | } |
| 26 | public string Инфо() |
| 27 | { |
| 28 | return string.Format("{0,-20}{1,-20}{2,10:f2}{3,10:d6}", |
| 29 | "Электроэнергия",фамилия,Сумма(),потребление); |
| 30 | } |
| 31 | } |
| 32 | class Запрос |
| 33 | { |
| 34 | public static void Заполнить(Электро[] платеж) |
| 35 | { |
| 36 | string фам; |
| 37 | int счПред=0, счТек=0; |
| 38 | Console.Clear(); |
| 39 | for (int i = 0; i < платеж.Length; i++) |
| 40 | { |
| 41 | Console.Write("Платеж "+ i + ": Фамилия -> "); |
| 42 | фам = Console.ReadLine(); |
| 43 | Console.Write("Платеж " + i + ": Текущее значение счетчика -> "); |
| 44 | счТек = int.Parse(Console.ReadLine()); |
| 45 | if (счТек > 0) |
| 46 | { |
| 47 | Console.Write("Платеж " + i + ": Предыдущее значение счетчика -> "); |

```
48      счПред = int.Parse(Console.ReadLine());
49      }
50      if(счТек <= 0)
51      платеж[i] = new Электро(фам);
52      else
53      платеж[i] = new Электро(фам, счТек, счПред);
54      }
55      }
56      public static void Вывести(Электро[] платеж)
57      {
58      for (int i = 0; i < платеж.Length; i++)
59      Console.WriteLine(платеж[i].Инфо());
60      }
61      }
62      class Платежи
63      {
64      static void Main(string[] args)
65      {
66      ConsoleKeyInfo rep;
67      Электро[] плэ;
68      int кпэ;
69      do
70      {
71      Console.Clear();
72      Console.Write("Количество платежей за электроэнергию: ");
73      кпэ = int.Parse(Console.ReadLine());
74      плэ = new Электро[кпэ];
75      Запрос.Заполнить(плэ);
76      Запрос.Вывести(плэ);
77      Console.WriteLine("Для выхода нажмите ESC");
78      rep = Console.ReadKey(true);
79      }while(rep.Key!= ConsoleKey.Escape);
80      }
81      }
82      }
```

*Рис. 1. Пример реализации программы «Платеж за электроэнергию»*

## Оценка характеристик программы

Рассмотрим текст программы для оценки ее качества с помощью метрик С. Чидамбера и К. Кемерера, которые позволяют оценить меру сложности объектно-ориентированной программы на основе анализа сложности методов классов программы и их связности, так как все классы работают во взаимодействии.

Определим метрику взвешенных методов на класс *wmc*, которая определяет сложность класса на основе анализа сложности

определенных в нем методов. Рассмотрим самый простой случай определения метрики сложности, когда в качестве критерия сложности принимается количество строк в исходном коде метода. Как отмечалось ранее, программа содержит в своем составе три класса. Проанализируем каждый из классов на предмет их сложности.

Класс Электро в своем составе имеет четыре метода:

public Электро (string фамилия);

public Электро (string фамилия, int текущее, int предыдущее);

public double Сумма ();

public string Инфо().

Одноименные методы Электро отличаются друг от друга количеством входных параметров. Метод public Электро (string фамилия) имеет исходный код в 5 строк. Метод public Электро (string фамилия, int текущее, int предыдущее) имеет в своем составе 5 строк. Метод public double Сумма () состоит из 4 строк. Исходный код метода public string Инфо () включает 5 строк. Тогда сложность класса Электро можно определить следующим образом:

$$WMC(\text{Электро}) = L[0] + L[1] + L[2] + L[3] = \\ = 5 + 5 + 4 + 5 = 19.$$

Класс Запрос имеет в своем составе два метода:

*public static void* Заполнить (Электро[ ] платеж);

*public static void* Вывести {Электро[ ] платеж).

Исходный код метода public static void Заполнить (Электро[] платеж) состоит из 21 строки. Исходный код метода *public static void* Вывести (Электро [] платеж) включает 5 строк. Сложность класса Запрос составляет

$$WMC(\text{Запрос}) = L[0] + L[1] = 21 + 5 = 26$$

Класс Платежи содержит всего лишь один метод static void Main (string[] args), который в своем составе имеет 17 строк. Сложность класса Платежи составляет:

$$WMC(\text{Платежи}) = L[0] = 17$$

Исходя из полученных результатов можно определенно заметить, что наиболее сложным является класс Запрос, так как в этом классе сосредоточены наиболее сложные методы.

Определим значение метрики $NM$ (количество методов на класс). При разработке исходного кода поставленной задачи принцип наследования не применялся, поэтому в коде программы отсутствуют базовые классы и их наследники. Для определения значения метрики $NM$ будем подсчитывать только методы текущих классов:

$NM$ (Электро) $=$ 4;
$NM$ (Запрос) $=$ 2;
$NM$ (Платежи) $=$ 1.

Исходя из показателей метрики $NM$, самым сложным является класс Электро, так как этот класс содержит наибольшее количество методов.

Определим связность между классами объектов CBO, которая численно определяется по количеству классов, с которыми связан анализируемый класс. Связь класса с другими классами определяется количеством обращений к вызову методов в других классах. В классе Электро осуществляется один вызов метода $string.Format$ из другого класса $string$ (см. рис. 1, строка 28), следовательно, CBO (Электро) $=$ 1.

В классе Запрос обращения к методам внешних классов осуществляется 12 раз:

- 8 раз - к различным методам класса $\boldsymbol{Console}$ (строки 38, 41, 42, 43,44, 47, 48 и 59);
- 2 раза - к методу $\boldsymbol{Parse}$ класса $\boldsymbol{int}$ (строки 44 и 48);
- 2 раза - к конструктору класса Электро (строки 51 и 53).

Таким образом, CBO (Запрос) $=$ 12.

В классе Платежи обращение к методам внешних классов осуществляется 10 раз:

- 5 раз - к методам класса $Console$ (строки 71, 72, 73, 77 и 78);
- 1 раз - к методу $Parse$ класса $int$ (строка 73);
- 1 раз - к конструктору класса Электро (строка 74);
- 2 раза - к методам класса Запрос (строки 75 и 76);
- 1 раз - к свойству класса $ConsoleKey$ (строка 79).

Таким образом, CBO (Платежи) $=$ 10.

Из полученных результатов значений метрик CBO следует, что наименее чувствительным к модификации является класс Электро, так как уровень связности его объектов с другими объектами является наименьшим. Уровни связности классов Запрос и Платежи относительно велики, что говорит об усложнении процессов модификации и тестирования всей программы в целом.

Определим количество откликов на класс $\boldsymbol{RFC}$, которое определяется числом методов, которые могут быть вызваны в рассматриваемом классе. Значение метрики $\boldsymbol{RFC}$ определяется по следующему соотношению:

$$RFC = R + M,$$

где $R$ - это методы, вызываемые методами рассматриваемого класса, M- общее количество методов класса.

Как уже рассматривалось ранее, класс Электро содержит четыре метода (M= 4), в одном из них вызываются методы $string.Format$ () и Сумма () ($R = 2$). Таким образом, метрика $RFC$ равна:

$$RFC = (\text{Электро}) = R + M = 2 + 4 = 6$$

Класс Запрос содержит два метода, которые осуществляют вызов 12 внешних методов, следовательно, $R = 12, M = 2$.

$$RFC(\text{Запрос}) = 12 + 2 = 14$$

Класс Платежи содержит один метод, который осуществляет вызов 10 внешних методов, поэтому R=10, M=1.

$$RFC(\text{Платежи}) = R + M = 10 + 1 = 11$$

Сложность классов ***Запрос*** и ***Платежи*** почти в два раза выше сложности класса ***Электро,*** поэтому для тестирования вызываемых методов в классах ***Запрос*** и ***Платежи*** потребуется большее количество тестов, следовательно, и отладка этих классов займет большее количество времени.

Определим значение метрики отсутствия сцепления в методах ***LCOM.***

Числовое значение этой характеристики определяется разностью между количеством пар методов, которые не используют одни и те же переменные в классе, и количеством пар методов, которые используют одни и те же переменные класса. Начнем с класса ***Электро.*** Все четыре метода этого класса попарно обращаются к одинаковым переменным класса:

- методы public Электро (string фамилия) и public Электро (string фамилия, int текущее, int предыдущее) обращаются к переменной потребление;
- методы public Электро (string фамилия) и public double Сумма() обращаются к переменной потребление;
- методы public Электро (string фамилия, int текущее, int предыдущее) и public double Сумма() обращаются к переменной потребление;
- методы public Электро (string фамилия) и public string Инфо() обращаются к переменным потребление и фамилия;
- методы public Электро (string фамилия, int текущее, int предыдущее) и public string Инфо() обращаются к переменным потребление и фамилия;

- методы public double Сумма() и public string Инфо{) обращаются к переменной потребление.

Таким образом, в классе ***Электро*** нет ни одной пары методов, не использующих общих атрибутов класса, и насчитывается 6 пар методов, которые обращаются к одинаковым переменным.

$$LCOM = 0 - 6 = -6.$$

Если ***LCOM*** < 0, то значение этой метрики принимают равным 0. Следовательно, для ***Электро*** значение ***LCOM***= 0.

В классе ***Запрос*** содержится два метода. Эти методы не используют общие атрибуты класса по причине их отсутствия. Таким образом:

$$LCOM \text{ (Запрос)} = 1 - 0 = 1.$$

В классе ***Платежи*** имеется только один метод, следовательно, $\boldsymbol{LCOM} = 0$.

Анализируя значения метрик ***LCOM*** всех классов программы, можно отметить, что уровень связности методов в классах достаточно высок, что позволяет в полной мере обеспечить инкапсуляцию классов.

## Задача «Геометрия окружности и прямоугольника»

В предметной области «Геометрия» определены информационные объекты, описывающие понятия «Точка», «Окружность» «Прямоугольник» в виде классов. Классы «Окружность» и «Прямоугольник» являются наследниками класса «Точка». Каждый из классов имеет метод вывода параметров объекта. Следует иметь в виду, что метод вывода в базовом классе не виртуальный и не абстрактный. Методы вывода в наследниках переопределены. Вызываемый метод будет определяться типом ссылки.

Базовый класс «Точка» описан следующим образом:

- закрытые поля:

  координата по оси X (вещественная);

  координата по оси Y (вещественная);

- метод: вывод координат точки.

Класс «Окружность» задан следующими элементами:

- закрытые поля:

  координаты центра - унаследованные поля;

радиус (вещественный);

- метод: вывод координат центра и длины окружности.

Класс «Прямоугольник» задается так:

- закрытые поля:

координаты левого верхнего угла - унаследованные поля;

основание (вещественное);

высота (вещественная);

- метод: вывод координат левого верхнего угла и площади.

В основном классе необходимо сформировать список геометрических объектов разных типов в виде массива и вывести параметры объектов на экран. Разработать исходный код программы, по которому определить оценки характеристик программы на основе объектно-ориентированных метрик Мартина.

## Реализация программы

Текст программы на языке C# для реализации возможного алгоритма решения поставленной задачи представлен на рис. 2.

| Номера строк | Строки программы |
|---|---|
| 1 | namespace EX1_1 |
| 2 | { |
| 3 | class Точка |
| 4 | { |
| 5 | protected double x, y; |
| 6 | public Точка(double x, double y) |
| 7 | { |
| 8 | this.x = x; this.y = y; |
| 9 | } |
| 10 | public void Показать() |
| 11 | { |
| 12 | Console.WriteLine("Точка ({0:f2}:{1:f2})", x, y); |
| 13 | } |
| 14 | } |
| 15 | class Окружность: Точка |
| 16 | { |
| 17 | private double r; |
| 18 | public Окружность(double x, double y, double r): base(x, y) |
| 19 | { |
| 20 | this.r = r; |
| 21 | } |
| 22 | public new void Показать() |
| 23 | { |
| 24 | Console.WriteLine("Центр({0:f2}:{1:f2}) Длина={2:f2}",x,y,2*Math.PI*r); |
| 25 | } |
| 26 | } |
| 27 | |
| 28 | class Прямоугольник: Точка |
| 29 | { |
| 30 | private double a, h; |

```
31  public Прямоугольник(double x, double y, double a, double h): base(x, y)
32  {
33   this.a = a; this.h = h;
34  }
35  public new void Показать()
36  {
37   Console.WriteLine("Угол({0:f2}:{1:f2}) Площадь={2:f2}",x, y, a*h);
38  }
39  }
40  class Program
41  {
42  static void Main(string[] args)
43  {
44   Точка[] список = new Точка[6];
45   список[0] = new Точка(-1.0, -1.0);
46   список[1] = new Окружность(1.0, 1.0, 1.0);
47   список[2] = new Прямоугольник(10.0, 10.0, 10.0, 10.0);
48   список[3] = new Точка(-2.0, -2.0);
49   список[4] = new Окружность(3.0, 3.0, 3.0);
50   список[5] = new Прямоугольник(20.0, 20.0, 0.5, 0.5);
51   for (int i = 0; i < список.Length; i++) список[i].Показать();
52   Console.WriteLine();
53   for (int i = 0; i < список.Length; i++)
54   {
55   if (список[i] is Точка)
56    ((Точка)список[i]).Показать();
57   else
58    if (список[i] is Окружность)
59    ((Окружность)список[i]).Показать();
60    else
61    if (список[i] is Прямоугольник)
62     ((Прямоугольник)список[i]).Показать();
63   }
64   Console.WriteLine();
65   for (int i = 0; i < список.Length; i++)
66   switch (список[i].GetType().Name)
67   {
68   case "Точка": ((Точка)список[i]).Показать(); break;
69   case "Окружность": ((Окружность)список[i]).Показать(); break;
70   case "Прямоугольник": ((Прямоугольник)список[i]).Показать(); break;
71   }
72   }
73   }
74  }
```

*Рис. 2. Пример реализации программы «Геометрия»*

## Оценка характеристик программы

Рассмотрим текст программы для оценки ее качества с помощью метрик С. Чидамбера и К. Кемерера, которые позволяют оценить меру сложности объектно-ориентированной программы на основе анализа сложности методов классов программы и их связности, так как все классы работают во взаимодействии.

Определим метрику взвешенных методов на класс *wmc,* которая определяет сложность класса на основе сложности определенных в нем методов. Рассмотрим простейший случай определения метрики сложности, когда в качестве критерия сложности принимается количество строк в исходном коде метода. Исходный код программы включает четыре класса, которые можно отнести к одной категории классов, предназначенной для решения данной задачи, целью которой является определение геометрических фигур на основании исходных координат точек, определяющих место положения этих фигур. Категория включает следующие классы:

- class Точка - базовый класс, определяющий понятие точка, расположенная на координатной плоскости;
- class Окружность - производный класс от class Точка, определяющий понятие окружность с центром в точке, расположенной на координатной плоскости;
- class Прямоугольник - производный класс от class Точка, определяющий понятие прямоугольник, расположение которого на плоскости определяется координатами точки верхнего левого его угла;
- class Program - класс, в котором содержится основной метод обработки массивов объектов типа Точка, Окружность и Прямоугольник.

Класс Точка в своем составе содержит два метода: $Public$ Точка ($double\ x, double\ y$) и $Public\ void$ Показать().

$$NM(\text{Точка}) = 2$$

Исходный код $public$ Точка ($double$ x, $double$ y) метода состоит из 4 строк. Исходный код public void Показать () состоит также из 4 строк. Сложность класса Точка определяется следующим образом:

$$WMC(\text{Точка}) = L[0] + L[1] = 4 + 4 = 8$$

Класс Окружность имеет в своем составе методы public Окружность (double x, double y, double r): base (x, y) и public new void

Показать(), исходные коды которых соответственно составляют 4 и 4 строки.

$$NM(\text{Окружность}) = 2$$
$$WMC(\text{Окружность}) = L[0] + L[1] = 4 + 4 = 8$$

Класс Прямоугольник включает два метода: public Прямоугольник (double x, double y, double a, double h): base (x, y) и public new void Показать (), которые содержат 4 и 4 строки соответственно:

$$NM(\text{Прямоугольник}) = 2$$
$$WMC(\text{Прямоугольник}) = 4 + 4 = 8$$

Класс Program содержит один метод Main(), исходный код которого состоит из 31 строки. Тогда

$$NM(Program) = 1$$
$$WMC(Program) = 31$$

Наиболее сложным по метрике ***WMC*** является основной класс ***Program.*** Сложность остальных классов программы одинаково невысока.

В программе реализован принцип наследования. Код программы содержит один базовый ***(Точка)*** и два производных класса (***Окружность*** и ***Прямоугольник).*** Глубина дерева наследования ***DIT*** = 1, поскольку имеется только один класс-предок.

Количество классов-потомков ***NOC*** = 2. Усложнение объектов производных классов невелико.

Определим связность между классами объектов ***CBO,*** которая численно определяется по количеству классов, с которыми связан анализируемый класс. Связь класса с другими классами определяется количеством обращений к вызову методов в других классах. В классе ***Точка*** имеет место одно обращение к методу другого класса ***Console*** (строка 12). Следовательно, ***CBO (Точка)*** = 1.

В классе ***Окружность*** наблюдается два обращения к методам других классов ***Console*** и ***Math*** (строка 24). Следовательно, ***CBO (Окружность)*** = 2.

Исходный код класса ***Прямоугольник*** содержит одно обращение к внешнему методу класса ***Console*** (строка 37), отсюда ***CBO (Прямоугольник)*** = 1.

Исходный код класса ***Program*** практически состоит из обращений к внешним методам классов ***Точка, Окружность, Прямоугольник*** и ***Console.*** В коде насчитывается 16 обращений к внешним методам. Таким образом, ***CBO (Program)*** = 16.

Наиболее сложным по связности является основной класс ***Program,*** остальные классы имеют достаточно низкую связность объектов.

Определим количество откликов на класс ***RFC,*** которое определяется числом методов, которые могут быть вызваны в рассматриваемом классе. Значение метрики ***RFC*** определяется по следующему соотношению:

$$RFC = R + M$$

где ***R*** - это методы вызываемые методами рассматриваемого класса, ***M***- общее количество методов класса.

Для класса ***Точка*** $R = 1$ - количество вызываемых в классе методов (***CBO (Точка)*** $= 1$), $M = 2$ - общее количество методов класса ***(NM(Точка)*** $= 2$), следовательно,

$$RFC(\text{точка}) = 1 + 2 = 3$$

Для класса ***Окружность*** $R = 2$ - количество вызываемых в классе методов ***(CBO (Окружность)*** $= 2$), $M = 2$ - общее количество методов класса ***(NM (Окружность)*** $= 2$), отсюда

$$RFC(\text{окружность}) = 2 + 2 = 4$$

Для класса ***Прямоугольник*** $R = 1$ — количество вызываемых в классе методов ***(CBO (Прямоугольник)*** $= 2$), $M = 2$ - общее количество методов класса ***(NM (Прямоугольник)*** $= 2$), поэтому

$$RFC(\text{прямоугольник}) = 1 + 2 = 3$$

Для класса ***Program*** $R = 16$ - количество вызываемых в классе методов ***(CBO (Program)*** $= 2$), $M = 1$ - общее количество методов класса ***(NM (Program) = 2),*** следовательно,

$$RFC(program) = 16 + 1 = 17$$

Исходя из полученных значений метрик ***RFC,*** наиболее сложным является класс ***Program.*** Остальные классы программы имеют низкую сложность.

Определим значение метрики отсутствия сцепления в методах ***LCOM.*** Числовое значение метрики определяется разностью между количеством пар методов, которые не используют одни и те же переменные в классе, и количеством пар методов, которые используют одни и те же переменные класса.

Класс Точка. Методы, определенные в этом классе, попарно обращаются к одним и тем же переменным:

$$LCOM = 0 - 1 = -1$$

При отрицательном результате значение LCOM= 0.

Класс ***Окружность.*** Методы, определенные в этом классе, попарно обращаются к одним и тем же переменным, отсюда

$$LCOM = 0 - 1 = -1$$

поэтому на основе того же принципа считаем, что LCOM= 0.

Класс Прямоугольник. Методы, определенные в этом классе, попарно обращаются к одним и тем же переменным, поэтому

$$LCOM = 0 - 1 = -1$$

следовательно, считаем значение $LCOM = 0$.

Для класса $Program\ LCOM = 0$, так как в этом классе определен всего лишь один метод.

Из анализа метрик следует, что сложность используемых классов невысокая, связность методов в классах - высокая. Чувствительность программы к модификации классов не очень высока. В целом можно считать качество программы соответствующим среднему уровню.