Министерство образования и науки Российской Федерации

Севастопольский государственный университет

Кафедра ИС

Отчет

По дисциплине: “Рефакторинг программного обеспечения”

Лабораторная работа №4

“Рефакторинг программного кода. Упрощение вызовов методов”

Выполнил:

ст.гр. ИС/б-17-2

Долженко И.А.

Проверил:

Строганов В.А.

Севастополь

2020

1 ЦЕЛЬ РАБОТЫ

Исследовать эффективность рефакторинга программного кода за счет упрощения вызовов методов. Получить практические навыки упрощения вызовов методов при рефакторинге объектно-ориентированных программ.

2 ХОД РАБОТЫ

**1. Переименование метода**

Ниже представлен фрагмент кода, в котором находится метод, имя которого не раскрывает свое значение:

void Main(string[] args)

{

    Console.Write("Введите размерность матрицы: ");

    int size = Int32.Parse(Console.ReadLine());

    Console.WriteLine("Исходный массив:");

    double[,] arr = new double[size, size];

    Random rnd = new Random();

*for* (int i = 0; i < size; i++)

*for* (int j = 0; j < size; j++)

            arr[i, j] = rnd.NextFloat(-10, 10);

    Prnt(arr);

    Console.WriteLine("Результат:");

*for* (int i = 0, length = arr.GetLength(0); i < length; i++) {

*for* (int j = i + 1; j < length; j++)

        {

            double temp = arr[i, j];

            arr[i, j] = arr[j, i];

            arr[j, i] = temp;

        }

    }

    Prnt(arr);

}

void Prnt(double[,] arr)

{

*for* (int i = 0, length = arr.GetLength(0); i < length; i++)

    {

*for* (int j = 0; j < length; j++) {

            Console.Write(Math.Round(arr[i, j], 2) + "\t");

        }

        Console.WriteLine();

    }

}

Изменим имя метода “Prnt” на “PrintArray”:

void Main(string[] args)

{

    Console.Write("Введите размерность матрицы: ");

    int size = Int32.Parse(Console.ReadLine());

    Console.WriteLine("Исходный массив:");

    ...

**PrintArray**(arr);

    Console.WriteLine("Результат:");

    ...

**PrintArray**(arr);

}

void **PrintArray**(double[,] arr)

{

*for* (int i = 0, length = arr.GetLength(0); i < length; i++)

    {

*for* (int j = 0; j < length; j++) {

            Console.Write(Math.Round(arr[i, j], 2) + "\t");

        }

        Console.WriteLine();

    }

}

**2-3.** **Разделение запроса и модификатора** *+* **Удаление параметра**

Ниже представлен фрагмент кода на языке Java Script, в котором присутствует метод возвращающий массив диалогов и одновременно изменяющий состояние другого объекта:

*export* class DialogStore {

    dialogs = {

        direct: [],

        groups: []

    };

    readed = {

        direct: false,

        groups: false

    };

    getDialogsByType = type => {

*if* (type === "direct") {

*this*.readed[type] = true;

        }

*return* *this*.dialogs[type];

    };

}

Создадим два метода – один для запроса, другой для модификации:

*export* class DialogStore {

    dialogs = {

        direct: [],

        groups: []

    };

    readed = {

        direct: false,

        groups: false

    };

**getDialogsByType = type => {**

***return* *this*.dialogs[type];**

**};**

**readDialogs = type => {**

***if* (type === "direct") {**

***this*.readed[type] = true;**

**}**

**};**

}

Так как теперь функция прочитывания диалога применяется отдельно, то в ней присутствует лишняя проверка на тип диалога и соответствующий параметр. Применим метод “Удаление параметра”:

*export* class DialogStore {

    dialogs = {

        direct: [],

        groups: []

    };

    readed = {

        direct: false,

        groups: false

    };

    getDialogsByType = type => {

*return* *this*.dialogs[type];

    };

**readDialogs = () => {**

***this*.readed.direct = true;**

**};**

}

**4. Параметризация метода**

В качестве примера представлен фрагмент кода на языке Java Script, который содержит методы форматирования даты по разному формату:

*import* { format } *from* "date-fns";

*export* const getDateMonth = () => {

*return* format(new Date(), "M");

};

*export* const getDateDays = () => {

*return* format(new Date(), "dd");

};

*export* const getDateSeconds = () => {

*return* format(new Date(), "ss");

};

...

Создадим универсальную функцию, которая в качестве параметров будет принимать дату и строку с правилом форматирования:

*import* { format } *from* "date-fns";

*export* const formatDate = (date, type = "HH:mm") => {

*return* format(new Date(date), type);

};

**5. Замена параметра вызовом метода**

Есть метод, выполняющий разный код в зависимости от значения параметра:

private void Add(string type)

{

    string[] row = new string[] { textBoxSubscriber.Text, textBoxPhone.Text, textBoxHome.Text, textBoxSum.Text };

*if* (type == "hash") {

        sWatch.Start();

        hashTable.Insert(row[0], row[1]);

        sWatch.Stop();

        setTimeLog("Добавление в хеш-таблицу: ", sWatch.ElapsedTicks.ToString());

        sWatch.Reset();

    }

*if* (type == "main") {

        sWatch.Start();

        appendToFile(row);

        sWatch.Stop();

        setTimeLog("Добавление в основную таблицу: ", sWatch.ElapsedTicks.ToString());

        sWatch.Reset();

    }

    dataGridViewFile.Rows.Add(row);

}

Создадим отдельный метод для каждого значения параметра:

private void AddToHashTable()

{

    string[] row = new string[] { textBoxSubscriber.Text, textBoxPhone.Text, textBoxHome.Text, textBoxSum.Text };

    sWatch.Start();

    hashTable.Insert(row[0], row[1]);

    sWatch.Stop();

    setTimeLog("Добавление в хеш-таблицу: ", sWatch.ElapsedTicks.ToString());

    sWatch.Reset();

    dataGridViewFile.Rows.Add(row);

}

private void AddToMainTable()

{

    string[] row = new string[] { textBoxSubscriber.Text, textBoxPhone.Text, textBoxHome.Text, textBoxSum.Text };

    sWatch.Start();

    appendToFile(row);

    sWatch.Stop();

    setTimeLog("Добавление в основную таблицу: ", sWatch.ElapsedTicks.ToString());

    sWatch.Reset();

    dataGridViewFile.Rows.Add(row);

}

**6. Удаление метода установки значения**

В классе Student, представленном ниже, в конструкторе устанавливаются поля, которые больше никогда не изменятся:

public class Student

{

    private string name;

    private int group;

    private int mark;

    private string subject;

    private DateTime time;

    public Student(string name, int group, string subject, int mark, DateTime time)

    {

        setName(name);

        setGroup(group);

        setSubject(subject);

        setMark(mark);

        setTime(time);

    }

    public setName(name)

    {

*this*.name = name;

    }

    public setGroup(group)

    {

*this*.group = group;

    }

    public setSubject(subject)

    {

*this*.subject = subject;

    }

    public setMark(mark)

    {

*this*.mark = mark;

    }

    public setTime(time)

    {

*this*.time = time;

    }

    public string getName()

    {

*return* name;

    }

    public int getGroup()

    {

*return* group;

    }

    public int getMark()

    {

*return* mark;

    }

    public string getSubject()

    {

*return* subject;

    }

    public DateTime getTime()

    {

*return* time;

    }

}

Удалим методы, устанавливающие значения этих полей:

public class Student

{

    private string name;

    private int group;

    private int mark;

    private string subject;

    private DateTime time;

    public Student(string name, int group, string subject, int mark, DateTime time)

    {

***this*.name = name;**

***this*.group = group;**

        setSubject(subject);

        setMark(mark);

        setTime(time);

    }

    public setSubject(subject)

    {

*this*.subject = subject;

    }

    public setMark(mark)

    {

*this*.mark = mark;

    }

    public setTime(time)

    {

*this*.time = time;

    }

    public string getName()

    {

*return* name;

    }

    ...

}

**7. Сокрытие метода**

Ниже представлен фрагмент кода класса AVL, предназначенного для построения AVL дерева:

class AVL

{

    Node root;

    public void Add(int data)

    {

        Node newItem = new Node(data);

*if* (root == null)

        {

            root = newItem;

        }

*else*

        {

            root = RecursiveInsert(root, newItem);

        }

    }

    public Node RecursiveInsert(Node current, Node n)

    {

*if* (current == null)

        {

            current = n;

*return* current;

        }

*else* *if* (n.data < current.data)

        {

            current.left = RecursiveInsert(current.left, n);

            current = balance\_tree(current);

        }

*else* *if* (n.data > current.data)

        {

            current.right = RecursiveInsert(current.right, n);

            current = balance\_tree(current);

        }

*return* current;

    }

    public Node balance\_tree(Node current)

    {

        int b\_factor = balance\_factor(current);

*if* (b\_factor > 1)

        {

*if* (balance\_factor(current.left) > 0)

                current = RotateLL(current);

*else*

                current = RotateLR(current);

        }

*else* *if* (b\_factor < -1)

        {

*if* (balance\_factor(current.right) > 0)

                current = RotateRL(current);

*else*

                current = RotateRR(current);

        }

*return* current;

    }

    public int max(int l, int r)

    {

*return* l > r ? l : r;

    }

    public int getHeight(Node current)

    {

        int height = 0;

*if* (current != null)

        {

            int l = getHeight(current.left);

            int r = getHeight(current.right);

            int m = max(l, r);

            height = m + 1;

        }

*return* height;

    }

}

В данном фрагменте кода присутствует множество методов с публичной областью видимости, использующихся лишь в данном классе. Сделаем методы закрытыми:

class AVL

{

    Node root;

    public void Add(int data)

    {

        ...

    }

**private** Node RecursiveInsert(Node current, Node n)

    {

        ...

    }

**private** Node balance\_tree(Node current)

    {

        ...

    }

**private** int max(int l, int r)

    {

        ...

    }

**private** int getHeight(Node current)

    {

        ...

    }

}

ВЫВОДЫ

В ходе выполнения лабораторной работы была исследована эффективность рефакторинга программного кода за счет упрощения вызовов методов. Получены практические навыки упрощения вызовов методов при рефакторинге объектно-ориентированных программ.