Лабораторная работа № 2
«Рефакторинг программного кода. Перемещение функций между объектами»

Цель работы

Исследовать эффективность перемещения функций между объектами при рефакторинге программного кода. Получить практические навыки применения приемов рефакторинга объектно-ориентированных программ.

Постановка задачи

1. Выбрать фрагмент программного кода для рефакторинга.
2. Выполнить рефакторинг программного кода, применив не менее 7 приемов, рассмотренных в разделе 2.2.
3. Составить отчет, содержащий подробное описание каждого модифицированного фрагмента программы и описание использованного метода рефакторинга.

Ход работы

1. Перемещение метода (Move Method).

Код до рефакторинга:

```
class Product
{
public:
    Product(const string &name, double price)
        : name_(name), price_(price) {}

    string getName() const
    {
        return name_;
    }

    double getPrice() const
    {
        return price_;
    }

private:
    string name_;
    double price_;
};

class ShoppingCart
{
public:
    void addProduct(const Product &product)
    {
        products_.push_back(product);
    }

    double getTotalPrice() const
    {
        double totalPrice = 0.0;

        for (const auto &product : products_)
        {
            totalPrice += product.getPrice();
        }

        return totalPrice;
    }

    void setDiscountRate(double discountRate)
    {
        discountRate_ = discountRate;
    }

    double applyDiscount(const Product &product) const
    {
        return product.getPrice() * (1.0 - discountRate_);
    }

private:
    vector<Product> products_;
    double discountRate_;
};
```

Код после рефакторинга:

```
class Product
{
public:
    Product(const string &name, double price)
        : name_(name), price_(price) {}

    string getName() const
    {
        return name_;
    }

    double getPrice() const
    {
        return applyDiscount();
    }

    void setDiscountRate(double discountRate)
    {
        discountRate_ = discountRate;
    }

    double applyDiscount() const
    {
        return price_ * (1.0 - discountRate_);
    }

private:
    string name_;
    double price_;
    double discountRate_ = 0.0;
};

class ShoppingCart
{
public:
    void addProduct(const Product &product)
    {
        products_.push_back(product);
    }

    double getTotalPrice() const
    {
        double totalPrice = 0.0;

        for (const auto &product : products_)
        {
            totalPrice += product.getPrice();
        }

        return totalPrice;
    }

private:
    vector<Product> products_;
};
```

2. Перемещение поля (Move Field).

Код до рефакторинга:

```
class Product
{
public:
    Product(const string &name, double price)
```

```
        : name_(name), price_(price) {}

    string getName() const
    {
        return name_;
    }

    double getPrice() const
    {
        return price_;
    }

private:
    string name_;
    double price_;
};

class ShoppingCart
{
public:
    void addProduct(const Product &product)
    {
        products_.push_back(product);
    }

    double getTotalPrice() const
    {
        double totalPrice = 0.0;

        for (const auto &product : products_)
        {
            totalPrice += product.getPrice();
        }

        return totalPrice;
    }

    void setDiscountRate(double discountRate)
    {
        discountRate_ = discountRate;
    }

    double applyDiscount(const Product &product) const
    {
        return product.getPrice() * (1.0 - discountRate_);
    }

private:
    vector<Product> products_;
    double discountRate_;
};
```

Код после рефакторинга:

```
class Product
{
public:
    Product(const string &name, double price)
```

```
        : name_(name), price_(price) {}

    string getName() const
    {
        return name_;
    }

    double getPrice() const
    {
        return applyDiscount();
    }

    void setDiscountRate(double discountRate)
    {
        discountRate_ = discountRate;
    }

    double applyDiscount() const
    {
        return price_ * (1.0 - discountRate_);
    }

private:
    string name_;
    double price_;
    double discountRate_ = 0.0;
};

class ShoppingCart
{
public:
    void addProduct(const Product &product)
    {
        products_.push_back(product);
    }

    double getTotalPrice() const
    {
        double totalPrice = 0.0;

        for (const auto &product : products_)
        {
            totalPrice += product.getPrice();
        }

        return totalPrice;
    }

private:
    vector<Product> products_;
};
```

3. Выделение класса (Extract Class).

Код до рефакторинга:

```
      class Employee
      {
      public:
          Employee() {}
          Employee(const string &name, const string &position, int salary, const
string &email)
            : m_name(name), m_position(position), m_salary(salary), m_email(email)
{}

          void setName(const string &name) { m_name = name; }
          string getName() const { return m_name; }

          void setPosition(const string &position) { m_position = position; }
          string getPosition() const { return m_position; }

          void setSalary(int salary) { m_salary = salary; }
          int getSalary() const { return m_salary; }

          void setEmail(const string &email) { m_email = email; }
          string getEmail() const { return m_email; }

          string getInfo() const
          {
              return "Name: " + m_name + ", Position: " + m_position + ", Salary:
" + to_string(m_salary) + ", Email: " + m_email;
          }

      private:
          string m_name;
          string m_position;
          int m_salary;
          string m_email;
      };
```

Код после рефакторинга:

```
      class ContactInfo
      {
      public:
          ContactInfo() {}
          ContactInfo(const string &email) : m_email(email) {}

          void setEmail(const string &email) { m_email = email; }
          string getEmail() const { return m_email; }

      private:
          string m_email;
      };

      class Employee
      {
      public:
          Employee() {}
          Employee(const string &name, const string &position, int salary, const
string &email)
              :     m_name(name),     m_position(position),     m_salary(salary),
m_contactInfo(email) {}
```

```
        void setName(const string &name) { m_name = name; }
        string getName() const { return m_name; }

        void setPosition(const string &position) { m_position = position; }
        string getPosition() const { return m_position; }

        void setSalary(int salary) { m_salary = salary; }
        int getSalary() const { return m_salary; }

        void   setContactInfo(ContactInfo   contactInfo)   {   m_contactInfo   =
contactInfo; }
        ContactInfo getContactInfo() const { return m_contactInfo; }

        string getInfo() const
        {
            return "Name: " + m_name + ", Position: " + m_position + ", Salary:
" + to_string(m_salary) + ", Email: " + m_contactInfo.getEmail();
        }

    private:
        string m_name;
        string m_position;
        int m_salary;
        ContactInfo m_contactInfo;
    };
```

4. Встраивание класса (Inline Class)

Код до рефакторинга:

```
    #include <iostream>
    #include <string>

    using namespace std;

    class Person {
    public:
       Person(const string& name, const string& street, const string& city, const
string& country)
            : name_(name), address_(street, city, country) {}

        void print() const {
            cout << "Name: " << name_ << endl;
            address_.print();
        }

    private:
        string name_;

        class Address {
        public:
          Address(const string& street, const string& city, const string& country)
                : street_(street), city_(city), country_(country) {}

            void print() const {
                cout << "Address:" << endl;
                cout << "Street: " << street_ << endl;
                cout << "City: " << city_ << endl;
                cout << "Country: " << country_ << endl;
            }

        private:
            string street_;
            string city_;
            string country_;
        };

        Address address_;
    };

    int main() {
        Person person("John Doe", "123 Main St", "New York", "USA");
        person.print();

        return 0;
    }
```

Код после рефакторинга:

```
#include <iostream>
#include <string>

using namespace std;

class Person {
public:
   Person(const string& name, const string& street, const string& city, const
string& country)
        : name_(name), street_(street), city_(city), country_(country) {}

    void print() const {
        cout << "Name: " << name_ << endl;
        cout << "Address:" << endl;
        cout << "Street: " << street_ << endl;
        cout << "City: " << city_ << endl;
        cout << "Country: " << country_ << endl;
    }

private:
    string name_;
    string street_;
    string city_;
    string country_;
};

int main() {
    Person person("John Doe", "123 Main St", "New York", "USA");
    person.print();

    return 0;
}
```

5. Сокрытие делегирования (Hide Delegate)

Код до рефакторинга:

```
#include <iostream>

using namespace std;

class Database
{
public:
    void connect(const string &host, const string &user, const string &password)
    {
        cout << "User: " << user << " connected to: " << host << " using password\n";
    }

    void disconnect()
    {
        cout << "User disconnected\n";
    }

    void saveUser(const string &name, const string &email)
    {
        cout << "User: " << name << " " << email << " saved to database\n";
    }
};

class User
{
private:
    Database database;

public:
    Database getDatabase()
    {
        return database;
    }

    void save(const string &name, const string &email)
    {
        database.saveUser(name, email);
    }
};

int main()
{
    User user;
    Database db = user.getDatabase();
    db.connect("localhost", "user", "password");
    user.save("John Doe", "john.doe@example.com");
    db.disconnect();
}
```

Код после рефакторинга:

```
#include <iostream>
```

```
    using namespace std;

    class Database
    {
    public:
        void connect(const string &host, const string &user, const string
&password)
        {
            cout << "User: " << user << " connected to: " << host << " using
password\n";
        }

        void disconnect()
        {
            cout << "User disconnected\n";
        }

        void saveUser(const string &name, const string &email)
        {
            cout << "User: " << name << " " << email << " saved to database\n";
        }
    };

    class User
    {
    private:
        Database database;

    public:
        void connect(const string &host, const string &user, const string
&password)
        {
            database.connect(host, user, password);
        }

        void disconnect()
        {
            database.disconnect();
        }

        void save(const string &name, const string &email)
        {
            database.saveUser(name, email);
        }
    };

    int main()
    {
        User user;
        user.connect("localhost", "user", "password");
        user.save("John Doe", "john.doe@example.com");
        user.disconnect();
    }
```

6. Удаление посредника (Remove Middle Man).

Код до рефакторинга:

```
#include <iostream>

class DrawingAPI
{
public:
    void drawCircle(int x, int y, int radius)
    {
        std::cout << "API.circle at " << x << "," << y << " with radius " << radius << "\n";
    };
    virtual void drawSquare(int x, int y, int side)
    {
      std::cout << "API.square at " << x << "," << y << " with side " << side << "\n";
    };
   virtual void drawTriangle(int x1, int y1, int x2, int y2, int x3, int y3)
    {
        std::cout << "API.triangle with points (" << x1 << "," << y1 << "), (" << x2 << "," << y2 << "), (" << x3 << "," << y3 << ")\n";
    };
};

class Shape
{
protected:
    DrawingAPI *drawingAPI;

public:
    Shape(DrawingAPI *drawingAPI) : drawingAPI(drawingAPI) {}
    virtual void draw() = 0;
};

class Circle : public Shape
{
private:
    int x, y, radius;

public:
    Circle(DrawingAPI *drawingAPI, int x, int y, int radius) : Shape(drawingAPI), x(x), y(y), radius(radius) {}

    void draw() override
    {
        drawingAPI->drawCircle(x, y, radius);
    }
};

class Square : public Shape
{
private:
    int x, y, side;

public:
  Square(DrawingAPI *drawingAPI, int x, int y, int side) : Shape(drawingAPI), x(x), y(y), side(side) {}
```

```
    void draw() override
    {
        drawingAPI->drawSquare(x, y, side);
    }
};

class Triangle : public Shape
{
private:
    int x1, y1, x2, y2, x3, y3;

public:
    Triangle(DrawingAPI *drawingAPI, int x1, int y1, int x2, int y2, int x3,
int y3) : Shape(drawingAPI), x1(x1), y1(y1), x2(x2), y2(y2), x3(x3), y3(y3) {}

    void draw() override
    {
        drawingAPI->drawTriangle(x1, y1, x2, y2, x3, y3);
    }
};

int main() {
    DrawingAPI* drawingAPI = new DrawingAPI();

    Circle* circle = new Circle(drawingAPI, 10, 20, 15);
    Square* square = new Square(drawingAPI, 50, 60, 20);
    Triangle* triangle = new Triangle(drawingAPI, 100, 110, 120, 130, 140,
150);

    circle->draw();
    square->draw();
    triangle->draw();

    delete circle;
    delete square;
    delete triangle;
    delete drawingAPI;

    return 0;
}
```

Код после рефакторинга:

```
#include <iostream>

class DrawingAPI
{
public:
    void drawCircle(int x, int y, int radius)
    {
        std::cout << "API.circle at " << x << "," << y << " with radius " <<
radius << "\n";
    };
    virtual void drawSquare(int x, int y, int side)
    {
      std::cout << "API.square at " << x << "," << y << " with side " << side
<< "\n";
```

```
        };
       virtual void drawTriangle(int x1, int y1, int x2, int y2, int x3, int y3)
        {
            std::cout << "API.triangle with points (" << x1 << "," << y1 << "),
(" << x2 << "," << y2 << "), (" << x3 << "," << y3 << ")\n";
        };
    };

    class Circle
    {
    private:
        DrawingAPI *drawingAPI;
        int x, y, radius;

    public:
        Circle(DrawingAPI   *drawingAPI,   int   x,   int   y,   int   radius)   :
drawingAPI(drawingAPI), x(x), y(y), radius(radius) {}

        void draw()
        {
            drawingAPI->drawCircle(x, y, radius);
        }
    };

    class Square
    {
    private:
        DrawingAPI *drawingAPI;
        int x, y, side;

    public:
        Square(DrawingAPI   *drawingAPI,   int   x,   int   y,   int   side)   :
drawingAPI(drawingAPI), x(x), y(y), side(side) {}

        void draw()
        {
            drawingAPI->drawSquare(x, y, side);
        }
    };

    class Triangle
    {
    private:
        DrawingAPI *drawingAPI;
        int x1, y1, x2, y2, x3, y3;

    public:
        Triangle(DrawingAPI *drawingAPI, int x1, int y1, int x2, int y2, int x3,
int y3) : drawingAPI(drawingAPI), x1(x1), y1(y1), x2(x2), y2(y2), x3(x3), y3(y3) {}

        void draw()
        {
            drawingAPI->drawTriangle(x1, y1, x2, y2, x3, y3);
        }
    };


    int main() {
```

```
    DrawingAPI* drawingAPI = new DrawingAPI();

    Circle* circle = new Circle(drawingAPI, 10, 20, 15);
    Square* square = new Square(drawingAPI, 50, 60, 20);
    Triangle* triangle = new Triangle(drawingAPI, 100, 110, 120, 130, 140,
150);

    circle->draw();
    square->draw();
    triangle->draw();

    delete circle;
    delete square;
    delete triangle;
    delete drawingAPI;

    return 0;
}
```

7. Введение внешнего метода (Introduce Foreign Method)

Код до рефакторинга:

Код после рефакторинга:

8. Введение локального расширения (Introduce Local Extension)

Код до рефакторинга:

Код после рефакторинга:

Выводы

В ходе выполнения лабораторной работы была исследована эффективность перемещения функций между объектами при рефакторинге программного кода. Также были получены практические навыки применения приемов рефакторинга объектно-ориентированных программ.

Были применены некоторые из приведённых методов рефакторинга к коду. В результате такого рефакторинга было выяснено, что в большинстве случаев код стал более структурированным и логичным, более понятным для чтения.