**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.