Министерство науки и высшего образования РФ

Федеральное государственное бюджетное образовательное учреждение высшего образования

"Кубанский государственный технологический университет"

(ФГБОУ ВО "КубГТУ")

Институт КСиИБ

Кафедра ИСП

**ОТЧЁТ**

по лабораторной работе №8

в рамках освоения дисциплины

**«Рефакторинг и работа с унаследованным кодом»**

**Тема: «Рефакторинг и расширение функциональности с использованием поведенческих паттернов проектирования»**

Обучающегося группы 21-КБ-ПР1

Направление подготовки 09.03.04 Программная инженерия

Студент Кузнецов Вадим Алексеевич

Преподаватель Степанова Елизавета Владимировна

Краснодар  
2024

**Цель работы:**

Изучить техники рефакторинга с использованием поведенческих паттернов проектирования.

**Задание:**

1) Изменить процесс назначения бонусов и скидок в соответствии с предложенными в пункте 2.2 изменениями, применив описанные в теории методы рефакторинга.

2) Проверить согласованность результатов и корректность работы нового варианта программы с результатами старой версии.

**1) Создать новые классы для реализации паттерна стратегия для бонусов и реализовать их в соответствии с документацией.**

//---Интерфейс стратегии для бонуса

public interface IBonusStrategy

{

int CalculateBonus(int quantity, decimal price);

}

// Реализация стратегии для обычных товаров

public class RegularBonusStrategy : IBonusStrategy

{

public int CalculateBonus(int quantity, decimal price)

{

if (quantity > 2)

return (int)(quantity \* price \* 0.05m); // 5% от суммы покупки

return 0;

}

}

// Реализация стратегии для товаров со скидкой

public class SaleBonusStrategy : IBonusStrategy

{

public int CalculateBonus(int quantity, decimal price)

{

return (int)(quantity \* price \* 0.01m); // 1% от суммы покупки

}

}

// Реализация стратегии для специальных товаров

public class SpecialBonusStrategy : IBonusStrategy

{

public int CalculateBonus(int quantity, decimal price)

{

// Нет бонусов для специальных товаров

return 0;

}

}

// Реализация стратегии для новогоднего периода обычных товаров

public class NewYearRegularBonusStrategy : IBonusStrategy

{

public int CalculateBonus(int quantity, decimal price)

{

// 7% от суммы покупки при покупке от 5000 руб.

var total = quantity \* price;

if (total > 5000)

return (int)(total \* 0.07m);

if (quantity > 2)

return (int)(quantity \* price \* 0.05m); // 5% от суммы покупки в обычном случае

return 0;

}

}

**2) Создать новые классы для реализации паттерна стратегия для скидок и реализовать их в соответствии с документацией.**

//---Интерфейс стратегии для расчета скидок

public interface IDiscountStrategy

{

decimal CalculateDiscount(int quantity, decimal price);

}

// Реализация стратегии для обычных товаров

public class RegularDiscountStrategy : IDiscountStrategy

{

public decimal CalculateDiscount(int quantity, decimal price)

{

if (quantity > 2)

return (quantity \* price) \* 0.03m; // 3% скидка

return 0;

}

}

// Реализация стратегии для товаров со скидкой

public class SaleDiscountStrategy : IDiscountStrategy

{

public decimal CalculateDiscount(int quantity, decimal price)

{

if (quantity > 3)

return (quantity \* price) \* 0.01m; // 0.1% скидка

return 0;

}

}

// Реализация стратегии для специальных товаров

public class SpecialDiscountStrategy : IDiscountStrategy

{

public decimal CalculateDiscount(int quantity, decimal price)

{

if (quantity > 10)

return (quantity \* price) \* 0.005m; // 0.5% скидка

return 0;

}

}

// Реализация стратегии для новогоднего периода для товаров со скидкой

public class NewYearSaleDiscountStrategy : IDiscountStrategy

{

public decimal CalculateDiscount(int quantity, decimal price)

{

var total = quantity \* price;

if (total > 2000)

return total \* 0.03m; // 3% скидка при покупке на сумму свыше 2000 руб.

if (quantity > 3)

return (quantity \* price) \* 0.01m; // 0.1% скидка

return 0;

}

}

// Реализация стратегии для новогоднего периода для акционных товаров

public class NewYearSpecialDiscountStrategy : IDiscountStrategy

{

public decimal CalculateDiscount(int quantity, decimal price)

{

var total = quantity \* price;

if (total > 3000)

return total \* 0.05m; // 5% скидка при покупке на сумму свыше 3000 руб.

return 0;

}

}

**3) Внедрить в класс Goods поля типов BonusStrategy и DiscountStrategy и добавить возможность их передачи через конструктор.**

public class Goods

{

protected string title;

protected int priceCode;

protected IBonusStrategy bonusStrategy;

protected IDiscountStrategy discountStrategy;

public Goods(string title, IBonusStrategy bonusStrategy, IDiscountStrategy discountStrategy)

{

this.title = title;

this.bonusStrategy = bonusStrategy;

this.discountStrategy = discountStrategy;

}

public string getTitle()

{

return title;

}  
 }

**4) Изменить процесс определения скидки и бонуса путем вызова соответствующих методов классов BonusStrategy и DiscountStrategy.**

//---Метод для получения бонусов

public virtual int GetBonus(int quantity, decimal price)

{

return bonusStrategy.CalculateBonus(quantity, price);

}

//---Метод для получения скидки

public virtual decimal GetDiscount(int quantity, decimal price)

{

return discountStrategy.CalculateDiscount(quantity, price);

}

**5) Добавить тесты для проверки новых вариантов подсчета скидок и бонусов (На этом этапе воспользуемся подходом TDD (разработка через тестирование): сначала напишем тесты, а потом код).**

[Test]

public void NewYearDiscountAndBonusForRegularGoods()

{

// Предположим, что даты предновогодних недель учитываются внутри стратегий

string testData = "CustomerName: Test\r\n" +

"CustomerBonus: 10\r\n" +

"GoodsTotalCount: 1\r\n" +

"# ID: NAME TYPE(REG/SAL/SPO)\r\n" +

"1: Milk REG\r\n" +

"ItemsTotalCount: 1\r\n" +

"# ID: GID PRICE QTY\r\n" +

"1: 1 100 50"; // Покупка на 5000 рублей

using (StringReader sr = new StringReader(testData))

{

BillGenerator billGenerator = billFactoryYaml.CreateBill(sr, "NewYearsSettings.json");

var result = billGenerator.GenerateBill();

// Ожидаем, что скидка не применяется, но бонусы начисляются 7% от 5000

Assert.IsTrue(result.Contains("Сумма счета составляет 4840,00"));

Assert.IsTrue(result.Contains("Вы заработали 350 бонусных балов"));

}

}

[Test]

public void UsualDiscountAndBonusForRegularGoods()

{

// Предположим, что даты предновогодних недель учитываются внутри стратегий

string testData = "CustomerName: Test\r\n" +

"CustomerBonus: 10\r\n" +

"GoodsTotalCount: 1\r\n" +

"# ID: NAME TYPE(REG/SAL/SPO)\r\n" +

"1: Milk REG\r\n" +

"ItemsTotalCount: 1\r\n" +

"# ID: GID PRICE QTY\r\n" +

"1: 1 100 50"; // Покупка на 5000 рублей

using (StringReader sr = new StringReader(testData))

{

BillGenerator billGenerator = billFactoryYaml.CreateBill(sr, "RegularSettings.json");

var result = billGenerator.GenerateBill();

// Ожидаем, что скидка не применяется, но бонусы начисляются 7% от 5000

Assert.IsTrue(result.Contains("Сумма счета составляет 4840,00"));

Assert.IsTrue(result.Contains("Вы заработали 250 бонусных балов"));

}

}

[Test]

public void NewYearDiscountForSaleGoods()

{

string testData = "CustomerName: Test\r\n" +

"CustomerBonus: 10\r\n" +

"GoodsTotalCount: 1\r\n" +

"# ID: NAME TYPE(REG/SAL/SPO)\r\n" +

"1: Christmas\_Tree SAL\r\n" +

"ItemsTotalCount: 1\r\n" +

"# ID: GID PRICE QTY\r\n" +

"1: 1 2000 2"; // Покупка на 4000 рублей

using (StringReader sr = new StringReader(testData))

{

BillGenerator billGenerator = billFactoryYaml.CreateBill(sr, "NewYearsSettings.json");

var result = billGenerator.GenerateBill();

// Ожидаем скидку 3% на товары со скидкой при покупке более чем на 2000 руб.

Assert.IsTrue(result.Contains("Сумма счета составляет 3880,00")); // 4000 - 3% скидка

Assert.IsTrue(result.Contains("Вы заработали 40 бонусных балов")); // 1% от 4000

}

}

[Test]

public void NewYearDiscountForSpecialGoods()

{

string testData = "CustomerName: Test\r\n" +

"CustomerBonus: 10\r\n" +

"GoodsTotalCount: 1\r\n" +

"# ID: NAME TYPE(REG/SAL/SPO)\r\n" +

"1: New\_Year\_Candy SPO\r\n" +

"ItemsTotalCount: 1\r\n" +

"# ID: GID PRICE QTY\r\n" +

"1: 1 300 11"; // Покупка на 3300 рублей

using (StringReader sr = new StringReader(testData))

{

BillGenerator billGenerator = billFactoryYaml.CreateBill(sr, "NewYearsSettings.json");

var result = billGenerator.GenerateBill();

// Ожидаем скидку 5% на акционные товары при покупке более чем на 3000 руб.

Assert.IsTrue(result.Contains("Сумма счета составляет 3125,00")); // 3300 - 5% скидка

Assert.IsTrue(result.Contains("Вы заработали 0 бонусных балов")); // Нет бонусов для акционных товаров

}

}

**5) Изменить процесс подсчёта бонусов и скидок, отделив его от основанного цикла подсчета суммы, согласовав его с написанными ранее тестами.**

**6) Добавить поддержку файла конфигурации и описание стратегий в нем (формат описания раздела стратегий выбрать самостоятельно).**

using System.Text.Json;

namespace РРУК\_01

{

// Класс генерации чеков

public class BillFactory

{

private IFileSource readContent;

public BillFactory(IFileSource readContent)

{

this.readContent = readContent;

}

//---Метод для преобразования данных из файла

public BillGenerator CreateBill(TextReader sr, string config = "RegularSettings.json")

{

int strategyType;

var configPath = Path.Combine(AppDomain.CurrentDomain.BaseDirectory, config);

if (!File.Exists(configPath))

{

throw new FileNotFoundException("Конфигурационный файл не найден.", configPath);

}

string configJson = File.ReadAllText(configPath);

var settings = JsonSerializer.Deserialize<ConfigSettings>(configJson);

if(settings.Season== "NewYears")

{

strategyType = 1;

}

else

{

strategyType = 0;

}

readContent.SetSource(sr, strategyType);

// Чтение покупателя

Customer customer = readContent.GetCustomer();

IView view = new TxtView();

BillGenerator b = new BillGenerator(customer, view);

//Чтение количества продуктов

int goodsQty = readContent.GetGoodsCount();

Goods[] g = new Goods[goodsQty];

//Чтение каждого продукта

for (int i = 0; i < g.Length; i++)

{

g[i] = readContent.GetNextGood();

}

//Чтение количества товаров

int itemsQty = readContent.GetItemsCount();

//Чтение каждого товара

for (int i = 0; i < itemsQty; i++)

{

b.addGoods(readContent.GetNextItem(g));

}

return b;

}

}

//Класс для расспознования конфигураций

public class ConfigSettings

{

public string Season { get; set; }

}

}

**7) Изменить процесс создания экземпляра класса Bill и выполнять создание экземпляров классов стратегий с чтением значений процентов скидок из конфигурационного файла.**

//Класс отвечающий за определения типа файла

public class GoodsFactory

{

public Goods Create(string type, string title, int strategyType)

{

IBonusStrategy bonusStrategy;

IDiscountStrategy discountStrategy;

switch (strategyType)

{

case 0: // Обычные стратегии

switch (type)

{

case "REG":

bonusStrategy = new RegularBonusStrategy();

discountStrategy = new RegularDiscountStrategy();

break;

case "SAL":

bonusStrategy = new SaleBonusStrategy();

discountStrategy = new SaleDiscountStrategy();

break;

case "SPO":

bonusStrategy = new SpecialBonusStrategy();

discountStrategy = new SpecialDiscountStrategy();

break;

default:

throw new ArgumentException("Неизвестный тип товара");

}

break;

case 1: // Новогодние стратегии

switch (type)

{

case "REG":

bonusStrategy = new NewYearRegularBonusStrategy();

discountStrategy = new RegularDiscountStrategy(); // Предполагается, что для RegularGoods скидка остаётся такой же как в обычный период

break;

case "SAL":

bonusStrategy = new SaleBonusStrategy(); // Бонусы для товаров со скидкой остаются неизменными даже в новогодний период

discountStrategy = new NewYearSaleDiscountStrategy();

break;

case "SPO":

bonusStrategy = new SpecialBonusStrategy(); // Предполагается, что для SpecialGoods бонусы не изменяются

discountStrategy = new NewYearSpecialDiscountStrategy();

break;

default:

throw new ArgumentException("Неизвестный тип товара");

}

break;

default:

throw new ArgumentException("Неизвестный тип стратегии");

}

return new Goods(title, bonusStrategy, discountStrategy);

}

}

**8) Свёрнута иерархия классов товаров.**

**До:**

//Класс обычного продукат

public class RegularGoods : Goods

{

public RegularGoods(string title) : base(title)

{

}

//---Метод для получения бонусов

public override int GetBonus(int \_quantity, decimal \_price)

{

return (int)(\_quantity \* \_price \* (decimal)0.05);

}

//---Метод для получения скидки

public override decimal GetDiscount(int \_quantity, decimal \_price)

{

if (\_quantity > 2)

return (\_quantity \* \_price) \* (decimal)0.03; // 3%

return 0;

}

}

//Класс продукта распродажи

public class SaleGoods : Goods

{

public SaleGoods(string title) : base(title)

{

}

//---Метод для получения бонусов

public override int GetBonus(int \_quantity, decimal \_price)

{

return (int)(\_quantity \* \_price \* (decimal)0.01);

}

//---Метод для получения скидки

public override decimal GetDiscount(int \_quantity, decimal \_price)

{

if (\_quantity > 3)

return (\_quantity \* \_price) \* (decimal)0.01; // 0.1%

return 0;

}

}

//Класс специального продукат

public class SpecialGoods : Goods

{

public SpecialGoods(string title) : base(title)

{

}

//---Метод для получения бонусов

public override int GetBonus(int \_quantity, decimal \_price)

{

return 0;

}

//---Метод для получения скидки

public override decimal GetDiscount(int \_quantity, decimal \_price)

{

if (\_quantity > 10)

return (\_quantity \* \_price) \* (decimal)0.005; // 0.5%

return 0;

}

}

public class BillFactory

{

private IFileSource readContent;

public BillFactory(IFileSource readContent)

{

this.readContent = readContent;

}

//---Метод для преобразования данных из файла

public BillGenerator CreateBill(TextReader sr)

{

readContent.SetSource(sr);

// Чтение покупателя

Customer customer = readContent.GetCustomer();

IView view = new TxtView();

BillGenerator b = new BillGenerator(customer, view);

//Чтение количества продуктов

int goodsQty = readContent.GetGoodsCount();

Goods[] g = new Goods[goodsQty];

//Чтение каждого продукта

for (int i = 0; i < g.Length; i++)

{

g[i] = readContent.GetNextGood();

}

//Чтение количества товаров

int itemsQty = readContent.GetItemsCount();

//Чтение каждого товара

for (int i = 0; i < itemsQty; i++)

{

b.addGoods(readContent.GetNextItem(g));

}

return b;

}

}

//Класс отвечающий за определения типа файла

public class GoodsFactory

{

public Goods Create(string type, string title)

{

switch (type)

{

case "REG":

return new RegularGoods(title);

case "SAL":

return new SaleGoods(title);

case "SPO":

return new SpecialGoods(title);

default:

throw new ArgumentException("Неизвестный тип товара");

}

}

}

**После:**

//Класс отвечающий за определения типа файла

public class GoodsFactory

{

public Goods Create(string type, string title, int strategyType)

{

IBonusStrategy bonusStrategy;

IDiscountStrategy discountStrategy;

switch (strategyType)

{

case 0: // Обычные стратегии

switch (type)

{

case "REG":

bonusStrategy = new RegularBonusStrategy();

discountStrategy = new RegularDiscountStrategy();

break;

case "SAL":

bonusStrategy = new SaleBonusStrategy();

discountStrategy = new SaleDiscountStrategy();

break;

case "SPO":

bonusStrategy = new SpecialBonusStrategy();

discountStrategy = new SpecialDiscountStrategy();

break;

default:

throw new ArgumentException("Неизвестный тип товара");

}

break;

case 1: // Новогодние стратегии

switch (type)

{

case "REG":

bonusStrategy = new NewYearRegularBonusStrategy();

discountStrategy = new RegularDiscountStrategy(); // Предполагается, что для RegularGoods скидка остаётся такой же как в обычный период

break;

case "SAL":

bonusStrategy = new SaleBonusStrategy(); // Бонусы для товаров со скидкой остаются неизменными даже в новогодний период

discountStrategy = new NewYearSaleDiscountStrategy();

break;

case "SPO":

bonusStrategy = new SpecialBonusStrategy(); // Предполагается, что для SpecialGoods бонусы не изменяются

discountStrategy = new NewYearSpecialDiscountStrategy();

break;

default:

throw new ArgumentException("Неизвестный тип товара");

}

break;

default:

throw new ArgumentException("Неизвестный тип стратегии");

}

return new Goods(title, bonusStrategy, discountStrategy);

}

}

[Test]

public void NewYearDiscountAndBonusForRegularGoods()

{

// Предположим, что даты предновогодних недель учитываются внутри стратегий

string testData = "CustomerName: Test\r\n" +

"CustomerBonus: 10\r\n" +

"GoodsTotalCount: 1\r\n" +

"# ID: NAME TYPE(REG/SAL/SPO)\r\n" +

"1: Milk REG\r\n" +

"ItemsTotalCount: 1\r\n" +

"# ID: GID PRICE QTY\r\n" +

"1: 1 100 50"; // Покупка на 5000 рублей

using (StringReader sr = new StringReader(testData))

{

BillGenerator billGenerator = billFactoryYaml.CreateBill(sr, "NewYearsSettings.json");

var result = billGenerator.GenerateBill();

// Ожидаем, что скидка не применяется, но бонусы начисляются 7% от 5000

Assert.IsTrue(result.Contains("Сумма счета составляет 4840,00"));

Assert.IsTrue(result.Contains("Вы заработали 350 бонусных балов"));

}

}

[Test]

public void UsualDiscountAndBonusForRegularGoods()

{

// Предположим, что даты предновогодних недель учитываются внутри стратегий

string testData = "CustomerName: Test\r\n" +

"CustomerBonus: 10\r\n" +

"GoodsTotalCount: 1\r\n" +

"# ID: NAME TYPE(REG/SAL/SPO)\r\n" +

"1: Milk REG\r\n" +

"ItemsTotalCount: 1\r\n" +

"# ID: GID PRICE QTY\r\n" +

"1: 1 100 50"; // Покупка на 5000 рублей

using (StringReader sr = new StringReader(testData))

{

BillGenerator billGenerator = billFactoryYaml.CreateBill(sr, "RegularSettings.json");

var result = billGenerator.GenerateBill();

// Ожидаем, что скидка не применяется, но бонусы начисляются 7% от 5000

Assert.IsTrue(result.Contains("Сумма счета составляет 4840,00"));

Assert.IsTrue(result.Contains("Вы заработали 250 бонусных балов"));

}

}

[Test]

public void NewYearDiscountForSaleGoods()

{

string testData = "CustomerName: Test\r\n" +

"CustomerBonus: 10\r\n" +

"GoodsTotalCount: 1\r\n" +

"# ID: NAME TYPE(REG/SAL/SPO)\r\n" +

"1: Christmas\_Tree SAL\r\n" +

"ItemsTotalCount: 1\r\n" +

"# ID: GID PRICE QTY\r\n" +

"1: 1 2000 2"; // Покупка на 4000 рублей

using (StringReader sr = new StringReader(testData))

{

BillGenerator billGenerator = billFactoryYaml.CreateBill(sr, "NewYearsSettings.json");

var result = billGenerator.GenerateBill();

// Ожидаем скидку 3% на товары со скидкой при покупке более чем на 2000 руб.

Assert.IsTrue(result.Contains("Сумма счета составляет 3880,00")); // 4000 - 3% скидка

Assert.IsTrue(result.Contains("Вы заработали 40 бонусных балов")); // 1% от 4000

}

}

[Test]

public void NewYearDiscountForSpecialGoods()

{

string testData = "CustomerName: Test\r\n" +

"CustomerBonus: 10\r\n" +

"GoodsTotalCount: 1\r\n" +

"# ID: NAME TYPE(REG/SAL/SPO)\r\n" +

"1: New\_Year\_Candy SPO\r\n" +

"ItemsTotalCount: 1\r\n" +

"# ID: GID PRICE QTY\r\n" +

"1: 1 300 11"; // Покупка на 3300 рублей

using (StringReader sr = new StringReader(testData))

{

BillGenerator billGenerator = billFactoryYaml.CreateBill(sr, "NewYearsSettings.json");

var result = billGenerator.GenerateBill();

// Ожидаем скидку 5% на акционные товары при покупке более чем на 3000 руб.

Assert.IsTrue(result.Contains("Сумма счета составляет 3125,00")); // 3300 - 5% скидка

Assert.IsTrue(result.Contains("Вы заработали 0 бонусных балов")); // Нет бонусов для акционных товаров

}

}

//---Интерфейс стратегии для бонуса

public interface IBonusStrategy

{

int CalculateBonus(int quantity, decimal price);

}

// Реализация стратегии для обычных товаров

public class RegularBonusStrategy : IBonusStrategy

{

public int CalculateBonus(int quantity, decimal price)

{

return (int)(quantity \* price \* 0.05m);// 5% от суммы покупки

}

}

// Реализация стратегии для товаров со скидкой

public class SaleBonusStrategy : IBonusStrategy

{

public int CalculateBonus(int quantity, decimal price)

{

return (int)(quantity \* price \* 0.01m); // 1% от суммы покупки

}

}

// Реализация стратегии для специальных товаров

public class SpecialBonusStrategy : IBonusStrategy

{

public int CalculateBonus(int quantity, decimal price)

{

// Нет бонусов для специальных товаров

return 0;

}

}

// Реализация стратегии для новогоднего периода обычных товаров

public class NewYearRegularBonusStrategy : IBonusStrategy

{

public int CalculateBonus(int quantity, decimal price)

{

// 7% от суммы покупки при покупке от 5000 руб.

var total = quantity \* price;

if (total >= 5000)

return (int)(total \* 0.07m);

else if (quantity > 2)

return (int)(quantity \* price \* 0.05m); // 5% от суммы покупки в обычном случае

return 0;

}

}

//---Интерфейс стратегии для расчета скидок

public interface IDiscountStrategy

{

decimal CalculateDiscount(int quantity, decimal price);

}

// Реализация стратегии для обычных товаров

public class RegularDiscountStrategy : IDiscountStrategy

{

public decimal CalculateDiscount(int quantity, decimal price)

{

if (quantity > 2)

return (quantity \* price) \* 0.03m; // 3% скидка

return 0;

}

}

// Реализация стратегии для товаров со скидкой

public class SaleDiscountStrategy : IDiscountStrategy

{

public decimal CalculateDiscount(int quantity, decimal price)

{

if (quantity > 3)

return (quantity \* price) \* 0.01m; // 0.1% скидка

return 0;

}

}

// Реализация стратегии для специальных товаров

public class SpecialDiscountStrategy : IDiscountStrategy

{

public decimal CalculateDiscount(int quantity, decimal price)

{

if (quantity > 10)

return (quantity \* price) \* 0.005m; // 0.5% скидка

return 0;

}

}

// Реализация стратегии для новогоднего периода для товаров со скидкой

public class NewYearSaleDiscountStrategy : IDiscountStrategy

{

public decimal CalculateDiscount(int quantity, decimal price)

{

var total = quantity \* price;

if (total > 2000)

return total \* 0.03m; // 3% скидка при покупке на сумму свыше 2000 руб.

else if (quantity > 3)

return (quantity \* price) \* 0.01m; // 0.1% скидка

return 0;

}

}

// Реализация стратегии для новогоднего периода для акционных товаров

public class NewYearSpecialDiscountStrategy : IDiscountStrategy

{

public decimal CalculateDiscount(int quantity, decimal price)

{

var total = quantity \* price;

if (total > 3000)

return total \* 0.05m; // 5% скидка при покупке на сумму свыше 3000 руб.

return 0;

}

}

**Контрольные вопросы**

**1) Как выполняется свертывание иерархии классов?**

Путем устранения лишних уровней наследования, объединяя функциональность подклассов и суперклассов в один класс, или удаляя неиспользуемые классы. Это делается для упрощения структуры кода, уменьшения сложности и улучшения читаемости. Процесс может включать в себя перемещение методов и полей вверх или вниз по иерархии, удаление классов, которые не добавляют уникального поведения, или замену наследования делегированием.

**2) Для чего используется паттерн Strategy?**

Для определения семейства алгоритмов, инкапсуляции каждого из них и обеспечения их взаимозаменяемости. Это позволяет выбирать алгоритм независимо от объектов-клиентов, которые его используют. Паттерн Strategy применяется, когда есть несколько родственных классов, которые отличаются только поведением, или когда необходимо изменять поведение объектов на стадии выполнения, или же когда в коде присутствует множество условных операторов, выбор которых зависит от одного и того же типа условий.

**3) Опишите преимущества использования паттерна «стратегия» при необходимости динамического изменения поведения?**

* **Гибкость и расширяемость**: Легко добавлять новые стратегии и изменять существующие без изменения контекста, который их использует.
* **Уменьшение условных операторов**: Код становится более чистым и легче для понимания, так как избавляется от множества условных конструкций за счет использования полиморфизма.
* **Инкапсуляция алгоритмов**: Каждый алгоритм заключается в свой собственный класс, что улучшает модульность и помогает изолировать код, связанный с выполнением определенных задач.
* **Взаимозаменяемость поведения**: Позволяет динамически изменять алгоритмы поведения объектов во время выполнения программы, что делает систему более гибкой.
* **Принцип открытости/закрытости**: Система остается открытой для расширения (можно легко добавлять новые стратегии), но закрытой для изменений (не требуется изменять существующий код при добавлении новых алгоритмов).

**Заключение**

В результате выполнения лабораторной работы были изучены техники рефакторинга с использованием поведенческих паттернов проектирования.