## Министерство образования Республики Беларусь

Учреждение образования «Витебский государственный технологический университет»

Кафедра: информационные системы и автоматизация производства

**Расчётно-графическая работа**

**по курсу**

**Технологии программирования**

**Вариант №9**

**Выполнил:**

**студент группы 3ИТ-4**

**Макавчик К. В.**

**Проверил:**

**Казаков В. Е.**

**Витебск, 2019**

**Задания к расчётно-графической работе**

**по курсу**

**Технологии программирования**

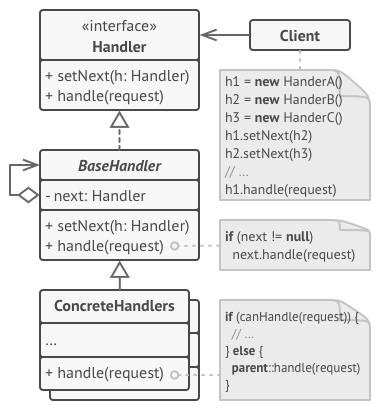
**Изучить и описать паттерн, согласно индивидуальному варианту.**

**Содержание отчёта:**

1. Описание паттерна: тип, задачи.
2. Диаграмма стандартной схемы организации паттерна
3. Описание назначения, задач и особенностей каждого из входящих в схему классов.
4. Преимущества применения паттерна.
5. Пример использования паттерна в разработке на языке С++ или Java.
6. Описание паттерна: тип, задачи.

Цепочка обязанностей — это поведенческий паттерн проектирования, который позволяет передавать запросы последовательно по цепочке обработчиков. Каждый последующий обработчик решает, может ли он обработать запрос сам и стоит ли передавать запрос дальше по цепи.

1. Диаграмма стандартной схемы организации паттерна



1. Описание назначения, задач и особенностей каждого из входящих в схему классов.

**Обработчик** определяет общий для всех конкретных обработчиков интерфейс. Обычно достаточно описать единственный метод обработки запросов, но иногда здесь может быть объявлен и метод выставления следующего обработчика.

**Базовый обработчик** — опциональный класс, который позволяет избавиться от дублирования одного и того же кода во всех конкретных обработчиках.

Обычно этот класс имеет поле для хранения ссылки на следующий обработчик в цепочке. Клиент связывает обработчики в цепь, подавая ссылку на следующий обработчик через конструктор или сеттер поля. Также здесь можно реализовать базовый метод обработки, который бы просто перенаправлял запрос следующему обработчику, проверив его наличие.

**Конкретные обработчики** содержат код обработки запросов. При получении запроса каждый обработчик решает, может ли он обработать запрос, а также стоит ли передать его следующему объекту.

В большинстве случаев обработчики могут работать сами по себе и быть неизменяемыми, получив все нужные детали через параметры конструктора.

**Клиент** может либо сформировать цепочку обработчиков единожды, либо перестраивать её динамически, в зависимости от логики программы. Клиент может отправлять запросы любому из объектов цепочки, не обязательно первому из них.

1. Преимущества применения паттерна и недостатки
2. уменьшает зависимость между клиентом и обработчиками.
3. реализует принцип единственной обязанности.
4. реализует принцип открытости/закрытости.
5. запрос может остаться никем не обработанным.
6. Пример использования паттерна в разработке на языке С++ или Java.

**class** Chain

Главный класс для обработки запрошенной пользователем суммы

**import** java.util.Scanner;

**package** Acvamarin.tp\_rgr\_chain;

**import** java.util.Scanner;

**public** **class** Chain {

**private** DispenseChain a1;

**public** Chain() {

**this**.a1 = **new** Dispense3();

DispenseChain a2 = **new** Dispense2();

DispenseChain a3 = **new** Dispense1();

a1.setNextChain(a2);

a2.setNextChain(a3);

}

**public** **static** **void** main(String[] args) {

Chain atmDispense = **new** Chain();

**while** (**true**) {

**int** amount = 0;

System.***out***.println("Введите сумму для выдачи:");

Scanner input = **new** Scanner(System.***in***);

amount = input.nextInt();

**if** (amount % 10 != 0) {

System.***out***.println("Сумма должна быть кратна 10-ки.");

**return**;

}

atmDispense.a1.dispense(**new** Currency(amount));

}

}

}

**class Currency**

Xранит сумму для распределения и использование цепных реализаций

**package** Acvamarin.tp\_rgr\_chain;

**public** **class** Currency {

**private** **int** amount;

**public** Currency(**int** amt) {

**this**.amount = amt;

}

**public** **int** getAmount() {

**return** **this**.amount;

}

}

**interface DispenseChain**

Имеет 2 метода:

1. для определения следующего процессора в цепочке
2. который обрабатывает запрос

**package** Acvamarin.tp\_rgr\_chain;

**public** **interface** DispenseChain {

**void** setNextChain(DispenseChain nextChain);

**void** dispense(Currency cur);

}

**class Dispense1**

Класс процессора, которые будут реализовывать DispenseChain интерфейс и обеспечивает реализацию методов dispense()

**package** Acvamarin.tp\_rgr\_chain;

**public** **class** Dispense1 **implements** DispenseChain {

**private** DispenseChain chain;

**public** **void** setNextChain(DispenseChain nextChain) {

**this**.chain = nextChain;

}

**public** **void** dispense(Currency cur) {

**if** (cur.getAmount() >= 10) {

**int** num = cur.getAmount() / 10;

**int** remainder = cur.getAmount() % 10;

System.***out***.println("Раздачи " + num + " 10$ ");

**if** (remainder != 0)

**this**.chain.dispense(**new** Currency(remainder));

} **else** {

**this**.chain.dispense(cur);

}

}

}

**class Dispense2**

Класс процессора, которые будут реализовывать DispenseChain интерфейс и обеспечивает реализацию методов dispense()

**package** Acvamarin.tp\_rgr\_chain;

**public** **class** Dispense2 **implements** DispenseChain {

**private** DispenseChain chain;

**public** **void** setNextChain(DispenseChain nextChain) {

**this**.chain = nextChain;

}

**public** **void** dispense(Currency cur) {

**if** (cur.getAmount() >= 20) {

**int** num = cur.getAmount() / 20;

**int** remainder = cur.getAmount() % 20;

System.***out***.println("раздачи " + num + " 20$ ");

**if** (remainder != 0)

**this**.chain.dispense(**new** Currency(remainder));

} **else** {

**this**.chain.dispense(cur);

}

}

}

**class Dispense3**

Класс процессора, которые будут реализовывать DispenseChain интерфейс и обеспечивает реализацию методов dispense()

**package** Acvamarin.tp\_rgr\_chain;

**public** **class** Dispense3 **implements** DispenseChain {

**private** DispenseChain chain;

**public** **void** setNextChain(DispenseChain nextChain) {

**this**.chain = nextChain;

}

**public** **void** dispense(Currency cur) {

**if** (cur.getAmount() >= 50) {

**int** num = cur.getAmount() / 50;

**int** remainder = cur.getAmount() % 50;

System.***out***.println("раздачи " + num + " 50$ ");

**if** (remainder != 0)

**this**.chain.dispense(**new** Currency(remainder));

} **else** {

**this**.chain.dispense(cur);

}

}

}