Class Adapter.java

```java
package src.main.java.base;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;


/* Паттерн Adapter
 *
 * Позволяет связать различные объекты без изменения их реализации.
 *
 * В моем случае это адаптер, который связывает компьютеры с сервером.
 * В данной реализации он берет от компьютеров только id и имя.
 * При увеличении сложности реализации можно добавить состояние, запросы
и так далее, что
 * более приблизит к реальной версии компьютерного адаптера.
 *
 *
 * JavaRush explain
 *
 * Используя паттерн, мы можем объединить два несовместимых объекта.
 * Конвертер между двумя несовместимыми объектами.
 *
*/




class Server {
    private Map<Integer, List<String>> messages = new HashMap<>();

    public void sendMessage(int fromId, int toId, String message) {
        System.out.println("Сервер: Компьютер " + fromId + " → Компьютер
" + toId + ": " + message);

        if (!messages.containsKey(toId)) {
            messages.put(toId, new ArrayList<>());
        }
        messages.get(toId).add("От " + fromId + ": " + message);
    }

    public List<String> getMessages(int computerId) {
        return messages.getOrDefault(computerId, new ArrayList<>());
    }
}
class Computer {
    private int id;
    private String name;

    public Computer(int id, String name) {
        this.id = id;
        this.name = name;
    }
    public int getId() { return id; }
    public String getName() { return name; }
}
public class Adapter {
    private Server server;

    public Adapter(Server server) {
```

```
        this.server = server;
    }
    public void sendMessage(Computer from, Computer to, String message) {
        server.sendMessage(from.getId(), to.getId(), message);
    }
    public void showMessages(Computer computer) {
        List<String> messages = server.getMessages(computer.getId());
        System.out.println("Сообщения для " + computer.getName() + ":");
        for (String msg : messages) {
            System.out.println("  " + msg);
        }
    }
}
```

Class Builder.java

```java
package src.main.java.base;

/* Паттерн Builder(класс Builder или же AppBuilder)
 *
 * Позволяет создавать различные конфигурации телефона с помощью одного класса.
 * По сути объединение сложных процессов в одном классе для компактности и сокращения
 * строк кода.
 *
 * Мой Builder выдает телефон, который состоит из модулей(модули так то тоже могут состоять из каких то других
 * классов или интерфейсов). Позволяет менять внутренние модули и сразу создавать собранный телефон.
 *
 *
 *
 * JavaRush explain
 *
 * Используется для создания сложного объекта с использованием простых объектов.
 * Постепенно он создает больший объект от малого и простого объекта.
 * Позволяет изменять внутреннее представление конечного продукта.
 *
*/


interface Module{
    void set(String name);
    void show();
}

class Software implements Module {
    String soft;
    public void set(String name){
        this.soft = name;
    }
    public void show(){
        System.out.println("Software: "+soft);
    }
}
class Case implements Module {
    String _case;
    public void set(String name){
        this._case = name;
    }
    public void show(){
        System.out.println("Case: "+_case);
    }
}
class CPU implements Module {
    String cpu;
    public void set(String name){
        this.cpu = name;
    }
    public void show(){
        System.out.println("CPU: "+cpu);
    }
}
class Battery implements Module {
    String battery;
```

```
    public void set(String name){
        this.battery = name;
    }
    public void show(){
        System.out.println("Battery: "+battery);
    }
}

class Phone {
    Module[] moduls = new Module[4];

        public void addSoftware(String soft){
        moduls[0] = new Software();
        moduls[0].set(soft);
    }
        public void addCase(String _case){
        moduls[1] = new Case();
        moduls[1].set(_case);
    }
        public void addCPU(String cpu){
        moduls[2] = new CPU();
        moduls[2].set(cpu);
    }
        public void addBattery(String battery){
        moduls[3] = new Battery();
        moduls[3].set(battery);
    }

public void show() {
    for (Module module : moduls) {
        if (module != null) {
            module.show();
        }
    }
}
}

public class Builder {//На самом деле лучше было назвать AppBuilder
    public Phone buildPhone(String soft, String _case, String cpu, String
battery){
        Phone phone = new Phone();
        phone.addSoftware(soft);
        phone.addCase(_case);
        phone.addCPU(cpu);
        phone.addBattery(battery);
        return phone;
    }
}
```

Class ChainOfResponsibility.java

```java
package src.main.java.base;

/*
 * Паттерн Chain Of Responsibility
 *
 * По сути позволяет вызывать альтернативные методы решения проблемы.
 * Или по другому позволяет ставить объекты в очередь на определенных условиях.
 *
 *
 * Java Rush explain
 *
 * Позволяет избежать жесткой зависимости отправителя запроса от его получателя,
 * при этом запрос может быть обработан несколькими объектами.
*/

interface Location{
    void changeLocation(Location loc);
    void tryConnect();
}
class CandaIP implements Location{
    Location loc = null;
    public void changeLocation(Location loc){
        this.loc = loc;
    }
    public void tryConnect() {
        System.out.println("Trying to connect Canada...");
            if (Math.random() > 0.5) { // 50% шанс успешного подключения
                System.out.println("Connected successfully!");
            } else if (loc != null) {
                loc.tryConnect();
            } else {
                System.out.println("All connections failed!");
        }
    }
}
class GehmanyIP implements Location{
    Location loc = null;
    public void changeLocation(Location loc){
        this.loc = loc;
    }
    public void tryConnect() {
        System.out.println("Trying to connect Gehmany...");
            if (Math.random() > 0.5) { // 50% шанс успешного подключения
                System.out.println("Connected successfully!");
            } else if (loc != null) {
                loc.tryConnect();
            } else {
                System.out.println("All connections failed!");
        }
    }
}



public class ChainOfResponsibility {// По сути VPN
    public static void main(String[] args){
        Location canda = new CandaIP();
```

```
            Location gehmany = new GehmanyIP();
            canda.changeLocation(gehmany);
            canda.tryConnect();
        }
    }
```

Class Decorator.java

```java
package src.main.java.base;

/*
 * Паттерн Decorator
 *
 * Позволяет добавлять к существующему классу функционал.
 * В моем случае добавим к дому интерьер, что поменяет описание и стоимость.
 *
 * Java Rush explain
 *
 * Добавляет новые функциональные возможности существующего объекта без привязки его структуры.
*/


class Home{
    float cost = 1;
    String show(){
        return "Простой дом, цена: " + cost;
    }
}

abstract class Decorator extends  Home{
    protected Home decoratedHome;
    String show(){
        return decoratedHome.show();
    }
}
class HomeWithInterior extends Decorator {
    public HomeWithInterior(Home home) {
        this.decoratedHome = home;
    }

    String show() {
        this.cost = (float) (this.cost*1.5);//Ничего не понял
        return decoratedHome.show() + " + интерьер, цена: " + (cost);
    }
}
```

Class Proxy.java

```java
package src.main.java.base;

/* Паттерн Proxy
 *
 * По сути оборачивает объект и меняет поведение.
 *
 *
 *
 * Java Rush explain
 *
 * Представляет объекты, которые могут контролировать другие объекты
перехватывая их вызовы. Можно перехватить вызов оригинального объекта.
 *
*/


interface Call{
    void showNumber(String number);
    String getNumber();
}

class PhoneCall implements Call{
    String number;
    public void showNumber(String num){
        System.out.println(number);
    }
    public String getNumber(){return number;}
}

public class Proxy implements Call{
    PhoneCall phoneCall;

    public Proxy(PhoneCall call){this.phoneCall = call;}

    public void showNumber(String number){
        if (!number.startsWith("+7")||!number.startsWith("8")){
            System.out.println("This is not Russia region, reccomend not
to answer!"); //Или написать что подозрение на спам, подозрительный
звонок и т.д.
        }
        phoneCall.showNumber(getNumber());
    }
    public String getNumber(){return phoneCall.getNumber();}
}
```

Interface Strategic.java

```java
package src.main.java.base;

/* Паттерн Strategy
 *
 * Позволяет выбрать поведение объекта. В моем случае анализ игры в шахматы.
 * Продолжить, сдаться или играть дальше.
 *
 *
 *
 * Java Rush explain
 *
 * Определяет ряд алгоритмов позволяя взаимодействовать между ними.
 * Алгоритм стратегии может быть изменен во время выполнения программы.
*/

interface Strategic{
    void doStrategy();
}

class DrawStategy implements Strategic{
    public void doStrategy(){
        System.out.println("Согласен на ничью!");
    }
    public int strategyPoint(){return 1;}
}
class GiveUpStategy implements Strategic{
    public void doStrategy(){
        System.out.println("Отличная партия. Я сдаюсь.");
    }
        public int strategyPoint(){return 2;}
}
class PushStategy implements Strategic{
    public void doStrategy(){
        System.out.println("Продолжаем!");
    }
    public int strategyPoint(){return 3;}
}

class CheesGamer {
    Strategic strategy;
    void showStrategy(){strategy.doStrategy();}
    void changeStrategy(Strategic newStrategy){strategy = newStrategy;}
}
```

Class TestPatterns.java

```java
package src.main.java.base;



public class TestPatterns {
    public static void main(String[] args) {
        System.out.println("=== ТЕСТИРОВАНИЕ ПАТТЕРНОВ ===");

        testAdapter();
        testBuilder();
        testChainOfResponsibility();
        testDecorator();
        testProxy();
        testStrategy();
    }

    private static void testAdapter() {
        System.out.println("\n--- Адаптер ---");
        Server server = new Server();
        Adapter adapter = new Adapter(server);

        Computer comp1 = new Computer(1, "Игровой ПК");
        Computer comp2 = new Computer(2, "Рабочий ноутбук");
        Computer comp3 = new Computer(3, "Сервер");

        adapter.sendMessage(comp1, comp2, "Привет! Как дела?");
        adapter.sendMessage(comp2, comp1, "Отлично! А у тебя?");
        adapter.sendMessage(comp1, comp3, "Нужна статистика");
        adapter.sendMessage(comp3, comp1, "Статистика готова");

        adapter.showMessages(comp1);
        adapter.showMessages(comp2);
        adapter.showMessages(comp3);
    }

    private static void testBuilder() {
        System.out.println("\n--- Строитель ---");
        Builder builder = new Builder();


        Phone budgetPhone = builder.buildPhone("Android 10", "Пластик",
"Snapdragon 450", "3000mAh");
        Phone flagshipPhone = builder.buildPhone("iOS 15", "Стекло", "A15
Bionic", "4500mAh");

        System.out.println("Бюджетный телефон:");
        budgetPhone.show();

        System.out.println("Флагманский телефон:");
        flagshipPhone.show();
    }

    private static void testChainOfResponsibility() {
        System.out.println("\n--- Цепочка обязанностей ---");
        CandaIP canada = new CandaIP();
        GehmanyIP germany = new GehmanyIP();

        canada.changeLocation(germany);
        System.out.println("Пытаемся подключиться через цепочку:");
```

```
        canada.tryConnect();
    }

    private static void testDecorator() {
        System.out.println("\n--- Декоратор ---");
        Home simpleHome = new Home();
        System.out.println(simpleHome.show());


        HomeWithInterior decoratedHome = new HomeWithInterior(new
Home());

        System.out.println(decoratedHome.show());
    }

    private static void testProxy() {
        System.out.println("\n--- Прокси ---");
        PhoneCall realCall = new PhoneCall();
        realCall.number = "+7**********"; // Российский номер

        Proxy callProxy = new Proxy(realCall);
        System.out.println("Российский номер:");
        callProxy.showNumber(realCall.number);


        PhoneCall foreignCall = new PhoneCall();
        foreignCall.number = "+44*********"; // Британский номер

        Proxy foreignProxy = new Proxy(foreignCall);
        System.out.println("Иностранный номер:");
        foreignProxy.showNumber(foreignCall.number);
    }

    private static void testStrategy() {
        System.out.println("\n--- Стратегия ---");
        CheesGamer player = new CheesGamer();

        player.changeStrategy(new PushStategy());
        player.showStrategy();

        player.changeStrategy(new DrawStategy());
        player.showStrategy();

        player.changeStrategy(new GiveUpStategy());
        player.showStrategy();
    }
}
```

Ссылка на гитхаб:
https://github.com/SokIvan/Third_Homework