МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

Кафедра информатики и систем управления

ОТЧЕТ

по лабораторной работе №3

по дисциплине

Шаблоны проектирования программного обеспечения

РУКОВОДИТЕЛЬ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_ Жевнерчук Д.В,\_\_

(подпись) (фамилия, и.,о.)

СТУДЕНТ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_Игнаков К. М.\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_Фомин Е. Н.\_\_\_

(подпись) (фамилия, и.,о.)

\_\_\_19-ИВТ-3\_\_\_\_\_\_\_\_

(шифр группы)

Работа защищена «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_

С оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Нижний Новгород 2021

Вариант 12.

Задание:

Разработайте объектно-ориентированную модель телефона.

Телефон обладает атрибутами: номер, баланс, вероятность поступления звонка.

С телефоном можно выполнить следующие операции: позвонить, ответить на звонок, завершить разговор, пополнить баланс.

Телефон может находиться в следующих состояниях: Ожидание, Звонок, Разговор, Заблокирован (баланс отрицательный).

Реализуйте приложение, демонстрирующее переход телефона между состояниями.

# Проектное решение

**Обоснование выбора паттернов**

При выполнении работы был использован Spring Framework. Spring Framework представляет собой контейнер внедрения зависимостей, с несколькими удобными слоями (например: доступ к базе данных, прокси, аспектно-ориентированное программирование, RPC, веб-инфраструктура MVC). Это все позволяет вам быстрее и удобнее создавать Java-приложения.

Внашем примере было задействовано аспектно-ориентированное программирование. Его цель заглючалась в том, чтобы логгировать все вызываемые методы класса Phone. Сделано это на примере работы оперционных систем и программых обеспечений — когда ОС или ПО выходят из строя из-за некорректной работы или любого другого перебоя, то система старается решить эту проблему по истории .log файлов либо отправляет отчет в этими файлами производителю данного продукта.

Таким же образом работает логгер в данном примере: что возвращает метод, где он расположен, его название и аргументы.

Основная идея в том, что телефон может находиться в одном из нескольких состояний, которые всё время сменяют друг друга. Набор этих состояний, а также переходов между ними, предопределён и конечен. Находясь в разных состояниях, программа может по-разному реагировать на одни и те же события, которые происходят с ней.

Паттерн Состояние предлагает создать отдельные классы для каждого состояния, в котором может пребывать объект, а затем вынести туда поведения, соответствующие этим состояниям.

Вместо того, чтобы хранить код всех состояний, первоначальный объект, называемый контекстом, будет содержать ссылку на один из объектов-состояний и делегировать ему работу, зависящую от состояния.

Благодаря тому, что объекты состояний будут иметь общий интерфейс, контекст сможет делегировать работу состоянию, не привязываясь к его классу. Поведение контекста можно будет изменить в любой момент, подключив к нему другой объект-состояние.

Производными классами для класса State (Состояние) являются:

1. BlockedState – телефон не может совершать исходящие вызовы. Выйти из этого состояния можно только пополнив баланс.
2. WaitingState – телефон находится в ожидании команды. Из этого состояния возможно перейти в состояние разговора (TalkingState), приняв входящий, или в состояние вызова (CallingState), позвонив на какой-то номер.
3. CallingState – телефон находится в состоянии исходящего вызова. В последствии (после ответа собеседника) переход в состояние разговора (TalkingState).
4. TalkingState – телефон находится в состоянии разговора. Сбросив вызов, можно перейти в состояние ожидания (WaitingState).

Для удобного перехода между состояниями телефон получает команду в консоли от пользователя.

Класс Phone создан для хранения информации о телефоне: баланс, номер телефона, состояние телефона.

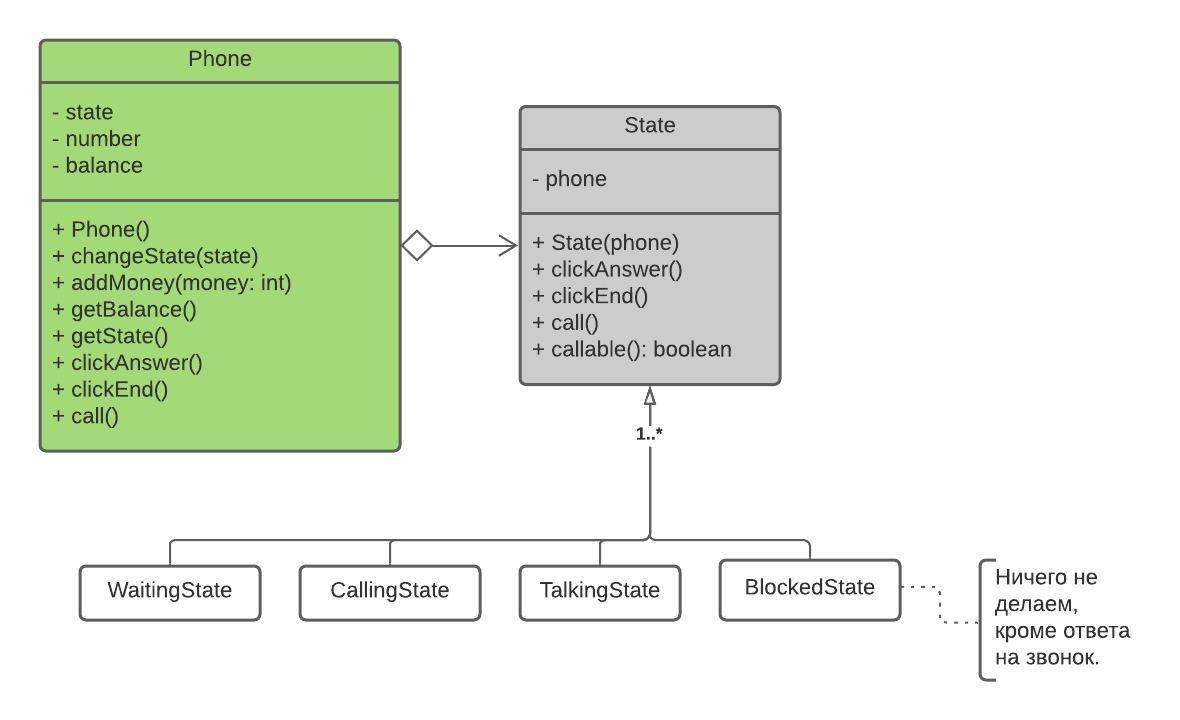


Рис. 1 – Диаграмма классов

# Приложение 1

**Программный код**

**main.java**

package com;

import com.phone.Phone;

import org.springframework.context.ApplicationContext;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import org.springframework.scheduling.annotation.EnableAsync;

import org.springframework.stereotype.Component;

import java.time.Duration;

import java.time.Instant;

import java.util.Scanner;

@Component

@EnableAsync

public class main {

public static void main(String[] args) {

ApplicationContext context = new AnnotationConfigApplicationContext("com");

Phone phone = context.getBean(Phone.class);

Scanner sc = new Scanner(System.in);

String command;

String state;

String help =

"\nhelp\t\t\t\tshow this text\n" +

"addMoney <float>\tadd money to the phone`s balance\n" +

"showBalance\t\t\tshow your phone`s balance\n" +

"answerCall\t\t\tanswer to incoming call\n" +

"call <number>\t\tcall to number (only if balance more then 0.5)\n" +

"endCall\t\t\t\tend talking\n" +

"turnOff\t\t\t\tturn off your phone\n";

String telephoneNumber;

float deposit = 0;

Instant start = Instant.now();

Instant finish;

System.out.println(help);

System.out.println("Phone is turn on!");

while (true) {

System.out.print("Enter command: ");

command = sc.next();

switch (command) {

case "showBalance":

System.out.println("Your balance: " + phone.getBalance());

break;

case "turnOff":

System.out.println("Turning off... Good bye!");

return;

case "addMoney":

deposit = sc.nextFloat();

if (phone.addMoney(deposit)) {

System.out.print("The balance is replenished!\n");

} else {

System.out.print("Something went wrong! " +

"May be you entered a negative amount or a zero?\n");

}

break;

case "call":

telephoneNumber = sc.next();

if (phone.call(telephoneNumber) == null) {

System.out.println("Can not call!");

} else {

start = Instant.now();

System.out.println("Calling on number: " + telephoneNumber);

}

break;

case "answerCall":

if (phone.clickAnswer() == null) {

System.out.println("Can not answer!");

} else {

System.out.println("Answer...");

start = Instant.now();

}

break;

case "endCall":

state = phone.clickEnd();

if (state == null) {

System.out.println("Can not end call!");

} else {

finish = Instant.now();

long elapsed = -(Duration.between(finish, start).toMillis()) / 1000;

System.out.println("Your phone conversation lasted: " + elapsed + " seconds");

System.out.println("Call is end.");

}

break;

case "help":

System.out.println(help);

break;

default:

System.out.println("You entered the wrong command\n" +

"Enter 'help' to get information about the program.");

break;

}

}

}

}

**AspectBuilder.java**

package com.AOP;

import org.aspectj.lang.JoinPoint;

import org.aspectj.lang.annotation.Aspect;

import org.aspectj.lang.annotation.Before;

import org.springframework.context.annotation.Configuration;

import org.springframework.context.annotation.EnableAspectJAutoProxy;

@Configuration

@Aspect

@EnableAspectJAutoProxy

public class AspectBuilder {

@Before("execution(\* com.phone.Phone.\*(..))")

public void AspectBefore(JoinPoint jp) {

System.out.println("LOGGER: " + jp.getSignature());

}

}

**Phone.java**

package com.phone;

import com.state.State;

import com.state.States.WaitingState;

import org.springframework.stereotype.Component;

@Component

public class Phone {

private State state;

private String number;

private float balance;

public Phone() {

this.state = new WaitingState(this);

}

public float getBalance() {

return balance;

}

public void changeState(State state) {

this.state = state;

}

public boolean addMoney(float money) {

if (money > 0) {

this.balance += money;

return true;

} else

return false;

}

public String clickAnswer() {

return this.state.clickAnswer();

}

public String clickEnd() {

return this.state.clickEnd();

}

public String call(String num) {

if (state.callable()) {

return this.state.call();

}

return null;

}

}

**State.java**

package com.state;

import com.phone.Phone;

import org.springframework.stereotype.Component;

@Component

public abstract class State {

public Phone phone;

public State(Phone phone) {

this.phone = phone;

}

public abstract boolean callable();

public abstract String clickAnswer();

public abstract String clickEnd();

public abstract String call();

}

**WaitingState.java**

package com.state.States;

import com.phone.Phone;

import com.state.State;

import org.springframework.stereotype.Component;

@Component

public class WaitingState extends State {

public WaitingState(Phone phone) {

super(phone);

}

/\*\*

\* Добавлено для удобства работы совершения звонков

\* Чтобы не обращаться к названию состояния и балансу, используем метод callable

\* Он проверяет возможность совершить звонок в данном состоянии телефона

\* Если баланс положительный, то мы можем позвонить из состояния ожидания

\*

\* @return boolean

\*/

@Override

public boolean callable() {

return phone.getBalance() > 0;

}

/\*\*

\* Мы можем ответить на входящий звонок в режиме ожидания

\*

\* @return String

\*/

@Override

public String clickAnswer() {

phone.changeState(new CallingState(phone));

return this.getClass().getName() + " -> clickAnswer" + " -> CallingState";

}

/\*\*

\* Проверяем баланс телефона

\* Если он положительный, то переходим к состоянию звонка,

\* Если он отрицательный, то переходим в режиму блокировки.

\*

\* @return String

\*/

@Override

public String call() {

if (phone.getBalance() > 0) {

phone.changeState(new CallingState(phone));

return this.getClass().getName() + " -> call" + " -> CallingState";

} else {

phone.changeState(new BlockedState(phone));

return this.getClass().getName() + " -> call" + " -> BlockedState";

}

}

/\*\*

\* В режиме ожидания мы не можем сбросить звонок

\*

\* @return null

\*/

@Override

public String clickEnd() {

return null;

}

}

**TalkingState.java**

package com.state.States;

import com.phone.Phone;

import com.state.State;

import org.springframework.stereotype.Component;

@Component

public class TalkingState extends State {

public TalkingState(Phone phone) {

super(phone);

}

/\*\*

\* Добавлено для удобства работы совершения звонков

\* Чтобы не обращаться к названию состояния и балансу, используем метод callable

\* Он проверяет возможность совершить звонок в данном состоянии телефона

\* Во время разговора мы не можем кому-то позвонить

\*

\* @return boolean

\*/

@Override

public boolean callable() {

return false;

}

/\*\*

\* Из состояния разговора нельзя ответить на ещё один звонок

\*

\* @return String

\*/

@Override

public String clickAnswer() {

return null;

}

/\*\*

\* Завершение звонка

\*

\* @return String

\*/

@Override

public String clickEnd() {

phone.changeState(new WaitingState(phone));

return this.getClass().getName() + " -> clickEnd" + " -> WaitingState";

}

/\*\*

\* Во время разговора мы не можем позвонить другому человеку

\*

\* @return String

\*/

@Override

public String call() {

return null;

}

}

**CallingState.java**

package com.state.States;

import com.phone.Phone;

import com.state.State;

import org.springframework.stereotype.Component;

@Component

public class CallingState extends State {

public CallingState(Phone phone) {

super(phone);

}

/\*\*

\* Добавлено для удобства работы совершения звонков

\* Чтобы не обращаться к названию состояния и балансу, используем метод callable

\* Он проверяет возможность совершить звонок в данном состоянии телефона

\* Во время вызова мы не можем позвонить ещё раз

\*

\* @return boolean

\*/

@Override

public boolean callable() {

return false;

}

/\*\*

\* В режиме разговора мы не можем ответить на звонок

\*

\* @return null

\*/

@Override

public String clickAnswer() {

return null;

}

/\*\*

\* В режиме разговора мы не можем позвонить

\*

\* @return null

\*/

@Override

public String call() {

return null;

}

/\*\*

\* Мы можем сбросить звонок при разговоре и перейти в режим ожидания

\*

\* @return String

\*/

@Override

public String clickEnd() {

phone.changeState(new WaitingState(phone));

return this.getClass().getName() + " -> clickEnd" + " -> WaitingState";

}

}

**BlockedState.java**

package com.state.States;

import com.phone.Phone;

import com.state.State;

import org.springframework.stereotype.Component;

@Component

public class BlockedState extends State {

public BlockedState(Phone phone) {

super(phone);

}

/\*\*

\* Добавлено для удобства работы совершения звонков

\* Чтобы не обращаться к названию состояния и балансу, используем метод callable

\* Он проверяет возможность совершить звонок в данном состоянии телефона

\* Если баланс положительный, то мы можем выйти из состояния блокировки и позвонить

\*

\* @return boolean

\*/

@Override

public boolean callable() {

return phone.getBalance() > 0;

}

/\*\*

\* Мы можем принять звонок от другого человека с отрицательным балансом

\*

\* @return String

\*/

@Override

public String clickAnswer() {

phone.changeState(new CallingState(phone));

return this.getClass().getName() + " -> clickAnswer" + " -> CallingState";

}

/\*\*

\* Мы ни с кем не разговариваем - не за чем завершать разговор.

\* Проверяем не стал ли баланс положительным для возможного перехода

\* к другому состоянию.

\*

\* @return String

\*/

@Override

public String clickEnd() {

if (phone.getBalance() > 0) {

phone.changeState(new WaitingState(phone));

return this.getClass().getName() + " -> clickEnd" + " -> WaitingState";

}

return null;

}

/\*\*

\* Проверяем не стал ли баланс положительным.

\* Если да, то переходим к состоянию звонка,

\* если нет, то ничего не делаем.

\*

\* @return String

\*/

@Override

public String call() {

if (phone.getBalance() > 0) {

phone.changeState(new CallingState(phone));

return this.getClass().getName() + " -> call" + " -> CallingState";

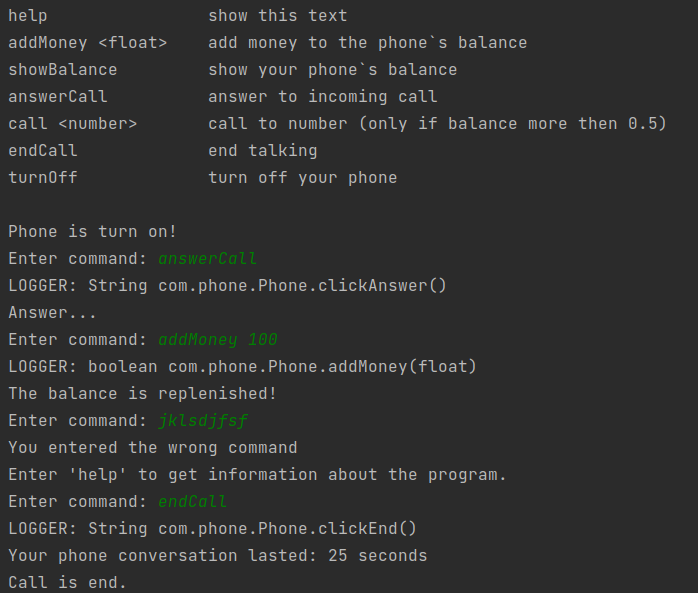
}

return null;

}

}

# Приложение 2

**Результаты тестирования**

