МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное учреждение высшего образования

«Ижевский государственный технический университет

имени М.Т. Калашникова»

Кафедра «Программное обеспечение»

**Отчет**

**по лабораторной работе №4**

**по дисциплине «Объектно-ориентированное программирование»**

Выполнил

студент группы Б22-191-2: Корепанов А.В.

Принял: старший преподаватель, Чернышев К.С.

Ижевск, 2023

**Постановка задачи**

В каждом приложении необходимо выделить классы, необходимые для описания предметной области; разработать структуру каждого класса (необходимые методы и поля). После этого реализуйте приложение с использованием разработанных типов данных. Всего необходимо реализовать не менее 3–4 классов в рамках автоматизации задачи.

Описание предметной области: “Пропускная система на предприятии”.

**Описание программы**

Наша программа состоит из 4 классов.

Рассмотрим 1 базовый класс – Employee. Он содержит поля о работнике (имя, фамилия, возраст, отдел, телефон), геттеры и сеттеры, а также дополнительные закрытые методы для проверки полей. Помимо этого, чтобы безопасно пользоваться hash коллекциями, необходимо переопределить методы equals и hashcode, для удобства вывода – toString. Данный класс можно было бы сделать record-ом, однако мы должны поддерживать возможность изменяемости полей класса.

Далее посмотрим на класс AccessPass. Он содержит поля (id, работник, список дат и времени входа на предприятие и выхода из него), а также геттеры и сеттеры для них. Помимо этого в классе есть 2 важных метода, которые вызываются при входе и выходе работника предприятия. Также, чтобы безопасно пользоваться hash коллекциями, необходимо переопределить методы equals и hashcode, для удобства вывода – toString.

Самый важный класс – AccessPassSystem. Он содержит множество AccessPass, а также некий counter, который позволяет автоинкрементровать пропуски новых работников. В данном классе определены методы, позволяющие добавить и удалить пропуск для конкретного работника. Другие 2 важных метода позволяют отметить, в какое время определенный работник вошел и вышел с работы. Также для удобства работников, отвечающих за пропускной режим, определены методы, позволяющие найти определенного работника, выдать всех работников, всех работников отдела. Самый интересный метод – метод, позволяющий определить количество минут, проведенных на работе определенным работником в период времени.

Последний класс – Main, в нем определена функция main, а также небольшие вспомогательные методы. В нем происходит вызов меню программы, а также выбор необходимого действия.

Также для каждого из классов, кроме Main, написаны JUnit тесты с помощью фреймворка JUnit 5. Скриншот покрытия тестов:

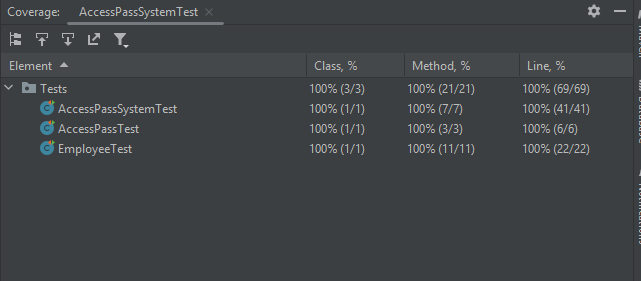


Рисунок 1. Покрытие тестами.

**Текст программы**

Ссылка на Git репозиторий:

https://github.com/alexkorep04/Labarotories-OOP

Класс Employee:

**package** Entity;

**import** java.util.Objects;

**import** java.util.regex.Matcher;

**import** java.util.regex.Pattern;

**public** **class** Employee {

**private** **String** name;

**private** **String** surname;

**private** int age;

**private** **String** department;

**private** **String** phone;

**public** Employee(**String** name, **String** surname, int age, **String** department, **String** phone) {

setName(name);

setSurname(surname);

setAge(age);

setDepartment(department);

setPhone(phone);

}

**public** **String** getName() {

**return** name;

}

**public** void setName(**String** name) {

**if**(checkNameSurnameDepartment(name)) {

**this**.name = name;

} **else** {

**throw** **new** **IllegalArgumentException**("Not correct value of name!");

}

}

**public** **String** getSurname() {

**return** surname;

}

**public** void setSurname(**String** surname) {

**if**(checkNameSurnameDepartment(surname)) {

**this**.surname = surname;

} **else** {

**throw** **new** **IllegalArgumentException**("Not correct value of surname!");

}

}

**public** int getAge() {

**return** age;

}

**public** void setAge(int age) {

**if**(checkAge(age)) {

**this**.age = age;

} **else** {

**throw** **new** **IllegalArgumentException**("Not correct value of age!");

}

}

**public** **String** getDepartment() {

**return** department;

}

**public** void setDepartment(**String** department) {

**if**(checkNameSurnameDepartment(department)) {

**this**.department = department;

} **else** {

**throw** **new** **IllegalArgumentException**("Not correct value of department!");

}

}

**public** **String** getPhone() {

**return** phone;

}

**public** void setPhone(**String** phone) {

**if**(checkPhone(phone)) {

**this**.phone = phone;

} **else** {

**throw** **new** **IllegalArgumentException**("Not correct value of phone!");

}

}

**@Override**

**public** boolean equals(**Object** o) {

**if** (**this** == o) **return** **true**;

**if** (o == **null** || getClass() != o.getClass()) **return** **false**;

Employee employee = (Employee) o;

**return** age == employee.age && **Objects**.equals(name, employee.name) && **Objects**.equals(surname, employee.surname) && **Objects**.equals(department, employee.department);

}

**@Override**

**public** int hashCode() {

**return** **Objects**.hash(name, surname, age, department);

}

**private** boolean checkNameSurnameDepartment(**String** string) {

**if**(string.length() >= 2) {

**for**(int i = 0; i < string.length(); i++) {

**if**(!**Character**.isLetter(string.charAt(i))) {

**return** **false**;

}

}

**return** **true**;

}

**return** **false**;

}

**private** boolean checkAge(int age) {

**return** age >= 18 && age <= 100;

}

**private** boolean checkPhone(**String** phone) {

**Pattern** pattern = **Pattern**.compile("\\+79\\d{9}");

**Matcher** matcher = pattern.matcher(phone);

**return** matcher.find();

}

**@Override**

**public** **String** toString() {

**return** "Employee{" +

"name='" + name + '\'' +

", surname='" + surname + '\'' +

", age=" + age +

", department='" + department + '\'' +

", phone='" + phone + '\'' +

'}';

}

}

Класс AccessPass:

**package** Access;

**import** Entity.Employee;

**import** java.time.LocalDateTime;

**import** java.util.ArrayList;

**import** java.util.List;

**import** java.util.Objects;

**public** **class** AccessPass {

**private** int passId;

**private** Employee employee;

**private** **List**<LocalDateTime> timesOfEntering;

**private** **List**<LocalDateTime> timesOfExiting;

**public** AccessPass(int passId, Employee employee) {

**this**.passId = passId;

**this**.employee = employee;

timesOfEntering = **new** **ArrayList**<>();

timesOfExiting = **new** **ArrayList**<>();

}

**public** int getPassId() {

**return** passId;

}

**public** void setPassId(int passId) {

**this**.passId = passId;

}

**public** Employee getEmployee() {

**return** employee;

}

**public** void setEmployee(Employee employee) {

**this**.employee = employee;

}

**public** **List**<LocalDateTime> getTimesOfEntering() {

**return** timesOfEntering;

}

**public** void setTimesOfEntering(**List**<LocalDateTime> timesOfEntering) {

**this**.timesOfEntering = timesOfEntering;

}

**public** **List**<LocalDateTime> getTimesOfExiting() {

**return** timesOfExiting;

}

**public** void setTimesOfExiting(**List**<LocalDateTime> timesOfExiting) {

**this**.timesOfExiting = timesOfExiting;

}

**public** void addEntering(LocalDateTime time) {

timesOfEntering.add(time);

}

**public** void addExiting(LocalDateTime time) {

timesOfExiting.add(time);

}

**@Override**

**public** boolean equals(**Object** o) {

**if** (**this** == o) **return** **true**;

**if** (o == **null** || getClass() != o.getClass()) **return** **false**;

AccessPass that = (AccessPass) o;

**return** passId == that.passId && **Objects**.equals(employee, that.employee) && **Objects**.equals(timesOfEntering, that.timesOfEntering) && **Objects**.equals(timesOfExiting, that.timesOfExiting);

}

**@Override**

**public** int hashCode() {

**return** **Objects**.hash(passId, employee, timesOfEntering, timesOfExiting);

}

**@Override**

**public** **String** toString() {

**return** "AccessPass{" +

"passId=" + passId +

", employee=" + employee +

", timesOfEntering=" + timesOfEntering +

", timesOfExiting=" + timesOfExiting +

'}';

}

}

Класс AccessPassSystem:

**package** Access;

**import** Entity.Employee;

**import** java.time.Duration;

**import** java.time.LocalDateTime;

**import** java.util.LinkedHashSet;

**import** java.util.List;

**import** java.util.Set;

**public** **class** AccessPassSystem {

**private** **Set**<AccessPass> passes;

**private** int curId;

**public** AccessPassSystem() {

passes = **new** **LinkedHashSet**<>();

curId = 1;

}

**public** void addEmployee(Employee employee) {

passes.add(**new** AccessPass(curId, employee));

curId++;

}

**public** void deleteEmployee(int id) {

AccessPass deletedPass = passes.stream().filter(pass -> pass.getPassId() == id).findFirst().get();

passes.remove(deletedPass);

}

**public** void enterToWork(int id) {

AccessPass accessPass = passes.stream().filter(pass -> pass.getPassId() == id).findFirst().get();

accessPass.addEntering(LocalDateTime.now());

}

**public** void exitToWork(int id) {

AccessPass accessPass = passes.stream().filter(pass -> pass.getPassId() == id).findFirst().get();

accessPass.addExiting(LocalDateTime.now());

}

**public** Employee getEmployeeByPassId(int id) {

**return** passes.stream().filter(pass -> pass.getPassId() == id).findFirst().get().getEmployee();

}

**public** **List**<Employee> getAllEmployees() {

**return** passes.stream().map(AccessPass::getEmployee).toList();

}

**public** **List**<Employee> getEmployeesByDepartment(**String** department) {

**return** passes.stream().map(AccessPass::getEmployee).filter(employee -> department.equals(employee.getDepartment())).toList();

}

**public** int getWorkingTimeBetweenDatesForEmployee(int id, LocalDateTime from, LocalDateTime to) {

AccessPass accessPass = passes.stream().filter(pass -> pass.getPassId() == id).findFirst().get();

**List**<LocalDateTime> enters = accessPass.getTimesOfEntering();

**List**<LocalDateTime> exits = accessPass.getTimesOfExiting();

int sumMinutes = 0;

**if**(exits.size() != enters.size()) {

**throw** **new** **IllegalArgumentException**("Not correct data!");

}

**for**(int i = 0; i < enters.size(); i++) {

sumMinutes += **Math**.abs(**Duration**.between(exits.get(i), enters.get(i)).toMinutes());

}

**return** sumMinutes;

}

**public** **Set**<AccessPass> getPasses() {

**return** passes;

}

}

Класс Main:

**import** Access.AccessPassSystem;

**import** Entity.Employee;

**import** java.time.LocalDate;

**import** java.time.LocalDateTime;

**import** java.time.format.DateTimeFormatter;

**import** java.util.Scanner;

**public** **class** Main {

**private** Main() {

}

**private** **static** **final** **Scanner** scanner = **new** **Scanner**(**System**.in);

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

**System**.out.flush();

AccessPassSystem accessPassSystem = **new** AccessPassSystem();

int choice = 1;

**while**(choice >= 1 && choice < 9) {

showMenu();

**String** name;

**String** surname;

int age;

int id;

**String** department;

**String** phone;

LocalDateTime to;

LocalDateTime from;

choice = scanner.nextInt();

**switch** (choice) {

**case** 1:

**System**.out.println("Введите имя работника:");

name = scanner.next();

**System**.out.println("Введите фамилию работника:");

surname = scanner.next();

**System**.out.println("Введите возраст:");

age = scanner.nextInt();

**System**.out.println("Введите отдел:");

department = scanner.next();

**System**.out.println("Введите номер телефона:");

phone = scanner.next();

accessPassSystem.addEmployee(**new** Employee(name, surname, age, department, phone));

**break**;

**case** 2:

**System**.out.println("Введите id пропуска работника:");

id = scanner.nextInt();

accessPassSystem.deleteEmployee(id);

**break**;

**case** 3:

**System**.out.println("Введите id пропуска работника:");

id = scanner.nextInt();

accessPassSystem.enterToWork(id);

**break**;

**case** 4:

**System**.out.println("Введите id пропуска работника:");

id = scanner.nextInt();

accessPassSystem.exitToWork(id);

**break**;

**case** 5:

**System**.out.println("Введите id пропуска работника:");

id = scanner.nextInt();

**System**.out.println(accessPassSystem.getEmployeeByPassId(id));

**break**;

**case** 6:

**System**.out.println(accessPassSystem.getAllEmployees());

**break**;

**case** 7:

**System**.out.println("Введите название отдела:");

department = scanner.next();

**System**.out.println(accessPassSystem.getEmployeesByDepartment(department));

**break**;

**case** 8:

**System**.out.println("Введите id пропуска работника:");

id = scanner.nextInt();

**System**.out.println("Введите дату и время в формате yyyy-MM-dd, с которой начинать отсчет:");

**String** userInput = scanner.next();

DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd");

from = LocalDate.parse(userInput, formatter).atStartOfDay();

**System**.out.println("введите дату и время в формате yyyy-MM-dd, которой завершать отсчет:");

userInput = scanner.next();

to = LocalDate.parse(userInput, formatter).atStartOfDay();

**System**.out.println(accessPassSystem.getWorkingTimeBetweenDatesForEmployee(id, from, to));

**break**;

**default**:

**break**;

}

**if**(choice > 0 && choice < 9) {

**System**.out.println("введите любой символ, чтобы выйти в меню");

**String** k = scanner.next();

}

}

}

**private** **static** void showMenu() {

**System**.out.println("�"обро пожаловать!");

**System**.out.println("Сделайте выбор.");

**System**.out.println("1. Добавить работника.");

**System**.out.println("2. Удалить работника.");

**System**.out.println("3. Указать, что работник пришел на работу.");

**System**.out.println("4. Указать, что работник ушел с работы.");

**System**.out.println("5. Получить работника по id.");

**System**.out.println("6. Получить всех работников предприятия.");

**System**.out.println("7. Получить работников из определенного отдела.");

**System**.out.println("8. Получить суммарное время работы работника за промежуток времени.");

**System**.out.println("0. закрыть программу");

}

}

Класс EmployeeTest:

**package** Tests;

**import** Entity.Employee;

**import** org.junit.jupiter.api.Assertions;

**import** org.junit.jupiter.api.BeforeEach;

**import** org.junit.jupiter.api.DisplayName;

**import** org.junit.jupiter.api.Test;

**import** **static** org.assertj.core.api.Assertions.assertThat;

**public** **class** EmployeeTest {

Employee employee;

**@BeforeEach**

**public** void createObject() {

employee = **new** Employee("Alexander", "Korepanov", 19, "IT", "+79120532635");

}

**@Test**

**@DisplayName("Test set name method, when we set normal name")**

**public** void testNormalName() {

Employee expected = **new** Employee("Nikita", "Korepanov", 19, "IT", "+79120532635");

employee.setName("Nikita");

assertThat(expected).isEqualTo(employee);

}

**@Test**

**@DisplayName("Test set name method, when we set not normal name")**

**public** void testNotNormalName() {

Assertions.assertThrows(**IllegalArgumentException**.class, () -> employee.setName("a"));

}

**@Test**

**@DisplayName("Test set surname method, when we set normal surname")**

**public** void testNormalSurname() {

Employee expected = **new** Employee("Alexander", "Kadukov", 19, "IT", "+79120532635");

employee.setSurname("Kadukov");

assertThat(expected).isEqualTo(employee);

}

**@Test**

**@DisplayName("Test set surname method, when we set not normal surname")**

**public** void testNotNormalSurname() {

Assertions.assertThrows(**IllegalArgumentException**.class, () -> employee.setSurname("a"));

}

**@Test**

**@DisplayName("Test set age method, when we set normal age")**

**public** void testNormalAge() {

Employee expected = **new** Employee("Alexander", "Korepanov", 20, "IT", "+79120532635");

employee.setAge(20);

assertThat(expected).isEqualTo(employee);

}

**@Test**

**@DisplayName("Test set ag method, when we set not normal age")**

**public** void testNotNormalAge() {

Assertions.assertThrows(**IllegalArgumentException**.class, () -> employee.setAge(101));

}

**@Test**

**@DisplayName("Test set department method, when we set normal department")**

**public** void testNormalDepartment() {

Employee expected = **new** Employee("Alexander", "Korepanov", 19, "Sales", "+79120532635");

employee.setDepartment("Sales");

assertThat(expected).isEqualTo(employee);

}

**@Test**

**@DisplayName("Test set department method, when we set not normal department")**

**public** void testNotNormalDepartment() {

Assertions.assertThrows(**IllegalArgumentException**.class, () -> employee.setDepartment("a"));

}

**@Test**

**@DisplayName("Test set phone method, when we set normal phone")**

**public** void testNormalPhone() {

Employee expected = **new** Employee("Alexander", "Korepanov", 19, "IT", "+79111111111");

employee.setPhone("+79111111111");

assertThat(expected).isEqualTo(employee);

}

**@Test**

**@DisplayName("Test set phone method, when we set not normal phone")**

**public** void testNotNormalPhone() {

Assertions.assertThrows(**IllegalArgumentException**.class, () -> employee.setPhone("89120532635"));

}

}

Класс AccessPassSystemTest:

**package** Tests;

**import** Access.AccessPass;

**import** Access.AccessPassSystem;

**import** Entity.Employee;

**import** org.junit.jupiter.api.BeforeEach;

**import** org.junit.jupiter.api.DisplayName;

**import** org.junit.jupiter.api.Test;

**import** java.time.LocalDateTime;

**import** java.util.List;

**import** **static** org.assertj.core.api.Assertions.assertThat;

**public** **class** AccessPassSystemTest {

AccessPassSystem accessPassSystem;

**@BeforeEach**

**public** void createObject() {

accessPassSystem = **new** AccessPassSystem();

}

**@Test**

**@DisplayName("Test add method")**

**public** void testAdd() {

AccessPass accessPass = **new** AccessPass(1, **new** Employee("Alexander", "Korepanov", 19, "IT", "+79120532635"));

accessPassSystem.addEmployee(**new** Employee("Alexander", "Korepanov", 19, "IT", "+79120532635"));

assertThat(accessPassSystem.getPasses()).contains(accessPass);

}

**@Test**

**@DisplayName("Test delete method")**

**public** void testDelete() {

AccessPass accessPass = **new** AccessPass(1, **new** Employee("Alexander", "Korepanov", 19, "IT", "+79120532635"));

accessPassSystem.addEmployee(**new** Employee("Alexander", "Korepanov", 19, "IT", "+79120532635"));

accessPassSystem.deleteEmployee(1);

assertThat(accessPassSystem.getPasses()).doesNotContain(accessPass);

}

**@Test**

**@DisplayName("Test employee by id method")**

**public** void testEmployeeById() {

AccessPass accessPass = **new** AccessPass(1, **new** Employee("Alexander", "Korepanov", 19, "IT", "+79120532635"));

accessPassSystem.addEmployee(**new** Employee("Alexander", "Korepanov", 19, "IT", "+79120532635"));

Employee expected = **new** Employee("Alexander", "Korepanov", 19, "IT", "+79120532635");

Employee response = accessPassSystem.getEmployeeByPassId(1);

assertThat(expected).isEqualTo(response);

}

**@Test**

**@DisplayName("Test get employees method")**

**public** void testEmployees() {

AccessPass accessPass1 = **new** AccessPass(1, **new** Employee("Alexander", "Korepanov", 19, "IT", "+79120532635"));

AccessPass accessPass2 = **new** AccessPass(2, **new** Employee("Nikita", "Korepanov", 19, "IT", "+79120532635"));

accessPassSystem.addEmployee(**new** Employee("Alexander", "Korepanov", 19, "IT", "+79120532635"));

accessPassSystem.addEmployee(**new** Employee("Nikita", "Korepanov", 19, "IT", "+79120532635"));

**List**<Employee> expected = **List**.of(**new** Employee("Alexander", "Korepanov", 19, "IT", "+79120532635"), **new** Employee("Nikita", "Korepanov", 19, "IT", "+79120532635") );

**List**<Employee> response = accessPassSystem.getAllEmployees();

assertThat(expected.size()).isEqualTo(response.size());

assertThat(expected).isEqualTo(response);

}

**@Test**

**@DisplayName("Test get employees by department method")**

**public** void testEmployeesByDep() {

AccessPass accessPass1 = **new** AccessPass(1, **new** Employee("Alexander", "Korepanov", 19, "IT", "+79120532635"));

AccessPass accessPass2 = **new** AccessPass(2, **new** Employee("Nikita", "Korepanov", 19, "IT", "+79120532635"));

AccessPass accessPass3 = **new** AccessPass(3, **new** Employee("Ivan", "Korepanov", 19, "Sales", "+79120532635"));

accessPassSystem.addEmployee(**new** Employee("Alexander", "Korepanov", 19, "IT", "+79120532635"));

accessPassSystem.addEmployee(**new** Employee("Nikita", "Korepanov", 19, "IT", "+79120532635"));

accessPassSystem.addEmployee(**new** Employee("Ivan", "Korepanov", 19, "Sales", "+79120532635"));

**List**<Employee> expected = **List**.of(**new** Employee("Alexander", "Korepanov", 19, "IT", "+79120532635"), **new** Employee("Nikita", "Korepanov", 19, "IT", "+79120532635") );

**List**<Employee> response = accessPassSystem.getEmployeesByDepartment("IT");

assertThat(expected.size()).isEqualTo(response.size());

assertThat(expected).isEqualTo(response);

}

**@Test**

**@DisplayName("Test get working time by employee")**

**public** void testWorkEmployees() {

Employee employee = **new** Employee("Alexander", "Korepanov", 19, "IT", "+79120532635");

accessPassSystem.addEmployee(employee);

accessPassSystem.getPasses().stream().findFirst().get().addEntering(LocalDateTime.of(2023, 1, 1, 12, 0 ,0));

accessPassSystem.getPasses().stream().findFirst().get().addExiting(LocalDateTime.of(2023, 1, 1, 12, 50 ,0));

accessPassSystem.getPasses().stream().findFirst().get().addEntering(LocalDateTime.of(2023, 1, 1, 17, 0 ,0));

accessPassSystem.getPasses().stream().findFirst().get().addExiting(LocalDateTime.of(2023, 1, 1, 20, 0 ,0));

int expected = 230;

int response = accessPassSystem.getWorkingTimeBetweenDatesForEmployee(1, LocalDateTime.MIN, LocalDateTime.MAX);

assertThat(expected).isEqualTo(response);

}

}

Класс AccessPassTest:

**package** Tests;

**import** Access.AccessPass;

**import** Entity.Employee;

**import** org.junit.jupiter.api.BeforeEach;

**import** org.junit.jupiter.api.DisplayName;

**import** org.junit.jupiter.api.Test;

**import** **static** org.assertj.core.api.Assertions.assertThat;

**import** java.time.LocalDateTime;

**public** **class** AccessPassTest {

AccessPass accessPass;

**@BeforeEach**

**public** void createObject() {

accessPass = **new** AccessPass(1, **new** Employee("Alexander", "Korepanov", 19, "IT", "+79120532635"));

}

**@Test**

**@DisplayName("Test add entering method")**

**public** void testEntering() {

accessPass.addEntering(LocalDateTime.MIN);

assertThat(accessPass.getTimesOfEntering()).contains(LocalDateTime.MIN);

}

**@Test**

**@DisplayName("Test add exiting method")**

**public** void testExiting() {

accessPass.addExiting(LocalDateTime.MIN);

assertThat(accessPass.getTimesOfExiting()).contains(LocalDateTime.MIN);

}

}

**Пример работы программы**

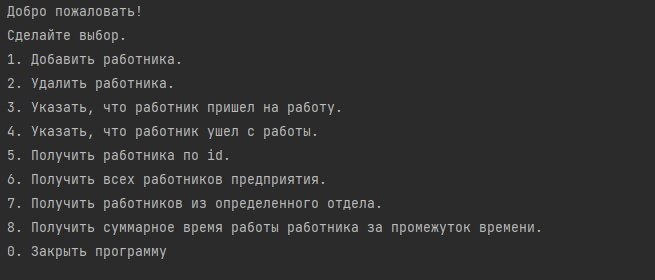


Рисунок 2. Главное меню.

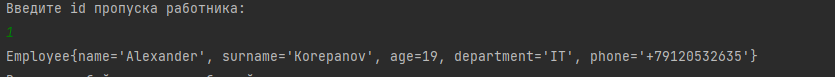


Рисунок 3. Вывод работника по id.



Рисунок 4. Вывод всех работников.

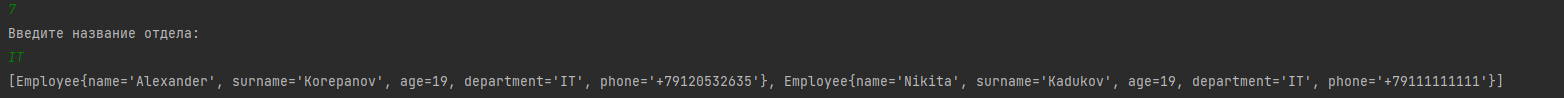


Рисунок 5. Вывод всех работников отдела.

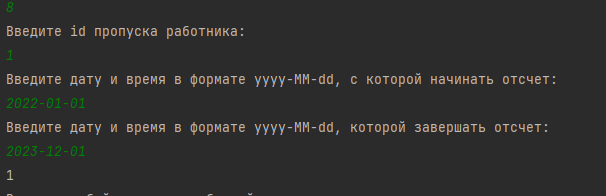


Рисунок 6. Вывод времени работы определенного работника.

**Вывод**

Таким образом, лабораторная работа была успешно выполнена на языке программирования Java в среде разработки IntelliJ IDEA Ultimate. Я повторил базовые понятия объектно-ориентированного программирования, потренировался в написании своих классов, создании конструкторов и методов, повторил принципы ООП, исключения, применил stream API и лямбды, повторил работу с датами. Также я написал свои юнит тесты для проверки работоспособности методов.