

**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ СВЯЗИ И МАССОВЫХ  
КОММУНИКАЦИЙ**

**Ордена Трудового Красного Знамени**

**Федеральное государственное бюджетное образовательное  
учреждение высшего образования**

**«Московский технический университет связи и информатики»**

Лабораторная работа № 2

«Создание иерархии классов»

Выполнила: Студентка группы БВТ2306

Максимова Мария

Москва 2024

Гитхаб:

<https://github.com/Mashaaaaa7/Lab2>

Для лабораторной работы №2 нам понадобится выполнить 1 задание:

### **Задание 1 Создайте иерархию классов в соответствии с вариантом.**

Вариант для сделанной лабораторной работы представляю такой:

Базовый класс: Сотрудник  
Дочерние классы: Администратор, Программист, Менеджер

Класс Main.java, в котором описаны все дочерние классы с информацией об имени, возрасте, заработной плате, опыте работы и языка программирования для программиста).

#### **1. Класс Main:**

- `public class Main`: определяет главный класс вашего приложения, который содержит метод `main()`.
- `public static void main(String[] args)`: метод `main()` - точка входа в вашу программу.

#### **2. Создание объектов:**

```
Programmist Programmist1 = new Programmist ("Максим", 30,25000,"Java");  
Administrator Administrator1 = new Administrator("Мария", 29, 25000, 5);  
Manager Manager1 = new Manager("Илья", 30, 25000,6);
```

- `Programmist Programmist1 = new Programmist("Максим", 30, 25000, "Java");`:: создает объект класса `Programmist` с именем `Programmist1` и передает ему имя, возраст, зарплату и язык программирования.
- `Administrator Administrator1 = new Administrator("Мария", 29, 25000, 5);`:: создает объект класса `Administrator` с именем `Administrator1` и передает ему имя, возраст, зарплату и количество опыта работы.

- `Manager Manager1 = new Manager("Илья", 30, 25000, 6);`: создает объект класса `Manager` с именем `Manager1` и передает ему имя, возраст, зарплату и количество опыта работы.

### 3. Вызов `displayInfo()`:

```
Programmist1.displayInfo();  
Administrator1.displayInfo();  
Manager1.displayInfo();  
}
```

- Эти строчки вызывают метод `displayInfo()` для каждого созданного объекта.

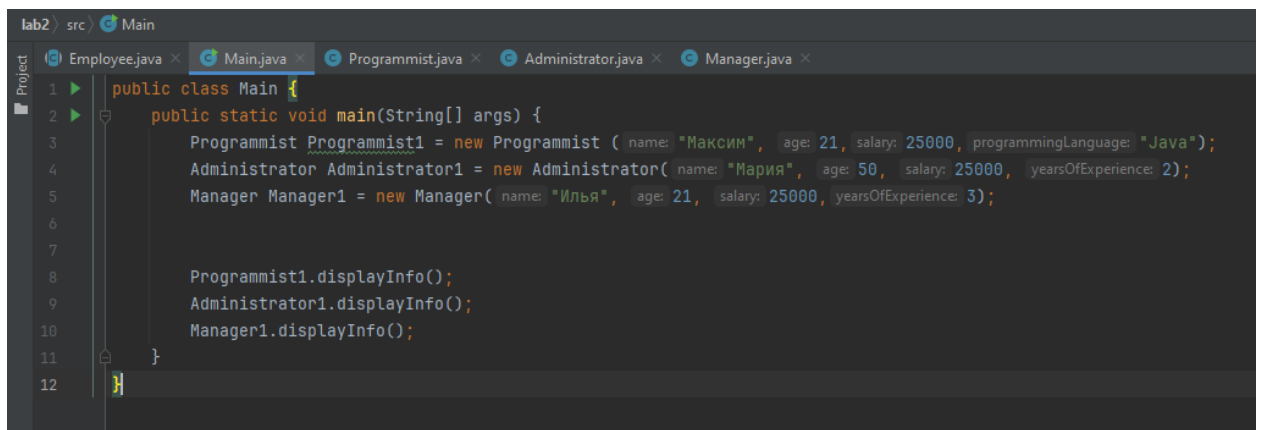


Рисунок 1

Абстрактный класс Сотрудник. Он базовый и выводит имя и возраст

```

1  public abstract class Employee { 3 usages 3 inheritors
2
3      private String name; 4 usages
4      private int age; 4 usages
5
6      public Employee (String name, int age) { 4 usages
7          this.name = name;
8          this.age = age;
9      }
10
11     public Employee (String name, String age) { this( name, "Мария", age, 20); }
12
13     public String getName() { return name; }
14
15     public void setName(String name) { this.name=name; // Устанавливаем новое значение для имени }
16
17     public int getAge() { no usages 3 overrides
18         return age;
19     }
20
21     public void setAge(int age) { this.age = age; }
22
23     public abstract void displayInfo(); // Абстрактный метод, который должен быть реализован в подклассах 3 usages 3 in
24
25     public String toString() { return "Имя: " + name + ", Возраст: " + age; }
26
27 }

```

Рисунок 2

Этот код представляет собой абстрактный класс Employee, который служит основой для представления сотрудников.

### 1. Абстрактный класс:

`public abstract class Employee { ... }` - объявляет абстрактный класс Employee. Это означает, что от него нельзя создать объект напрямую.

Абстрактные классы служат для создания шаблонов, которые должны быть расширены (наследоваться) другими классами.

### 2. Атрибуты (поля):

`private String name;` - хранит имя сотрудника.

`private int age;` - хранит возраст сотрудника.

`private` модификатор доступа делает эти поля доступными только внутри класса Employee.

### 3. Конструкторы:

`public Employee (String name, int age) { ... }` - конструктор, который инициализирует объект `Employee` с именем и возрастом.

`public Employee (String name, String age) { ... }` - этот конструктор некорректен. Он пытается создать объект `Employee` с помощью строки, но возраст должен быть целым числом.

#### 4. Методы доступа:

`public String getName() { ... }` - возвращает имя сотрудника.

`public void setName(String name) { ... }` - устанавливает новое значение для имени.

`public int getAge() { ... }` - возвращает возраст сотрудника.

`public void setAge(int age) { ... }` - устанавливает новый возраст.

#### 5. Абстрактный метод:

`public abstract void displayInfo();` - этот метод должен быть реализован в подклассах `Employee`, так как в базовом классе его реализация отсутствует. Он определяет общий интерфейс для вывода информации о сотруднике.

#### 6. `toString` метод:

`public String toString() { ... }` - переопределяет стандартный метод `toString` и возвращает строковое представление объекта `Employee` в виде "Имя: [имя], Возраст: [возраст]".

Класс Программиста: включает в себя реализованный публичный класс `Сотрудника`. Выводит имя, возраст, зарплату и язык программирования.

```

yee.java < Main.java < Programmist.java < Administrator.java < Manager.java <
public class Programmist extends Employee { 2 usages

    private String name; // имя 2 usages
    private int age; //возраст 2 usages
    private double salary; // зарплата 5 usages
    private String programmingLanguage; //язык программирования 5 usages

    public Programmist(String name, int age, double salary, String programmingLanguage) { 2 usages
        super(name, age); //аргументы name, age
        this.salary = salary;
        this.programmingLanguage = programmingLanguage;
    }

    public Programmist() { no usages
        this( name: "Максим", age: 20, salary: 25000, programmingLanguage: null);
    }

    // Геттеры и сеттеры
    public String getName() { no usages
        return name;
    }

    public void setName(String name) { no usages
        this.name = name;
    }
}

```

Рисунок 3

```

public int getAge() { no usages
    return age;
}

public void setAge(int age) { no usages
    this.age = age;
}

public double getSalary() { no usages
    return salary;
}

public void setSalary(double salary) { no usages
    this.salary = salary;
}

public String getProgrammingLanguage() { no usages
    return programmingLanguage;
}

public void setProgrammingLanguage(String programmingLanguage) { no usag
    this.programmingLanguage = programmingLanguage;
}

```

Рисунок 4

```

@Override

public String toString() {
    return "Программист: " + "Максим" + ", возраст:" + 21 + ", заработанная плата: " + salary + ", язык программирования: " + programmingLanguage;
}

public void displayInfo() { 3 usages
    System.out.println("Программист:" + "Максим" + ", возраст:" + 21 + ", Зарплата: " + salary + ", язык программирования: " + programmingLanguage);
}

```

Рисунок 5

## 1. Наследование от Employee:

```
public class Programmist extends Employee {
```

- public class Programmist: определяет класс Programmist, который является подклассом Employee.
- extends Employee: указывает, что класс Programmist наследует все свойства и методы от класса Employee.

## 2. Поля:

```
private String name;
```

```
private int age;
```

```
private double salary;
```

```
private String programmingLanguage;
```

- private: означает, что поля доступны только внутри класса Programmist.
- name: строка, хранящая имя программиста.
- age: целое число, хранящее возраст программиста.
- salary: вещественное число, хранящее зарплату программиста.
- programmingLanguage: строка, хранящая язык программирования, которым владеет программист.

## 3. Конструкторы:

```
public Programmist(String name, int age, double salary, String programmingLanguage) {
```

```
super(name, age);  
  
this.salary = salary;  
  
this.programmingLanguage = programmingLanguage;  
  
}
```

```
public Programmist() {  
  
    this("Максим", 30, 25000, null);  
  
}
```

### **Первый конструктор:**

- `public Programmist(String name, int age, double salary, String programmingLanguage):` инициализирует объект класса `Programmist` с указанными значениями для всех полей.
- `super(name, age);:` вызывает конструктор базового класса `Employee`, передавая ему имя и возраст.
- `this.salary = salary;:` присваивает значение поля `salary` параметру `salary`.
- `this.programmingLanguage = programmingLanguage;:` присваивает значение поля `programmingLanguage` параметру `programmingLanguage`.

### **Второй конструктор:**

- `public Programmist():` инициализирует объект класса `Programmist` со значениями по умолчанию.
- `this("Максим", 30, 25000, null);:` вызывает первый конструктор с указанными значениями по умолчанию.

## **4. Геттеры и Сеттеры:**



```
public String getName() { return name; }
```

```
public void setName(String name) { this.name = name; }
```

```
public int getAge() { return age; }
```

```
public void setAge(int age) { this.age = age; }
```

```
public double getSalary() { return salary; }
```

```
public void setSalary(double salary) { this.salary = salary; }
```

```
public String getProgrammingLanguage() { return programmingLanguage; }
```

```
public void setProgrammingLanguage(String programmingLanguage) {  
this.programmingLanguage = programmingLanguage; }
```

- Эти методы обеспечивают контролируемый доступ к значениям полей.

## 5. Переопределенный метод `toString()`:

```
@Override
```

```
public String toString() {
```

```
    return "Программист: " + "Максим" + ", возраст:" + 30 + ", заработанная  
плата: " + salary + ", язык программирования: " + programmingLanguage;  
}
```

- `@Override`: указывает, что этот метод переопределяет метод `toString()` из базового класса.
- Метод возвращает строку с информацией о программисте.

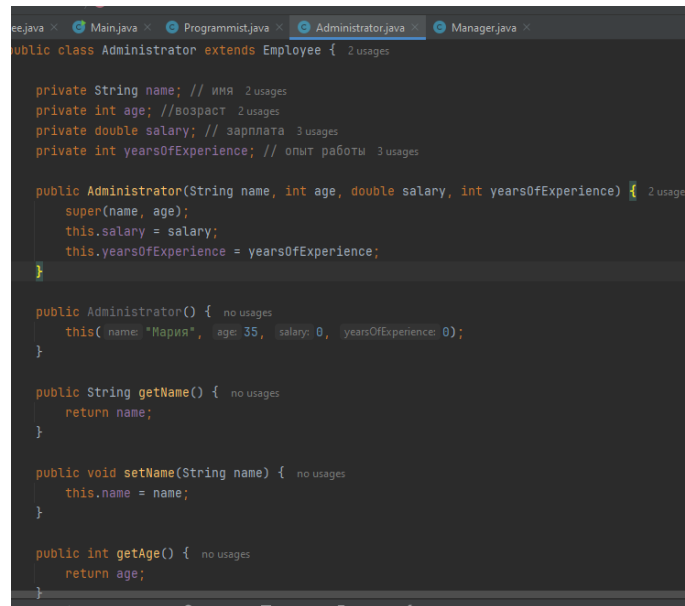
## 6. Реализация метода `displayInfo()`:

```
public void displayInfo() {
```

```
    System.out.println("Программист:" + "Максим" + ", возраст:" + 30 + ",  
Зарплата: " + salary + ", язык программирования: " + programmingLanguage);  
}
```

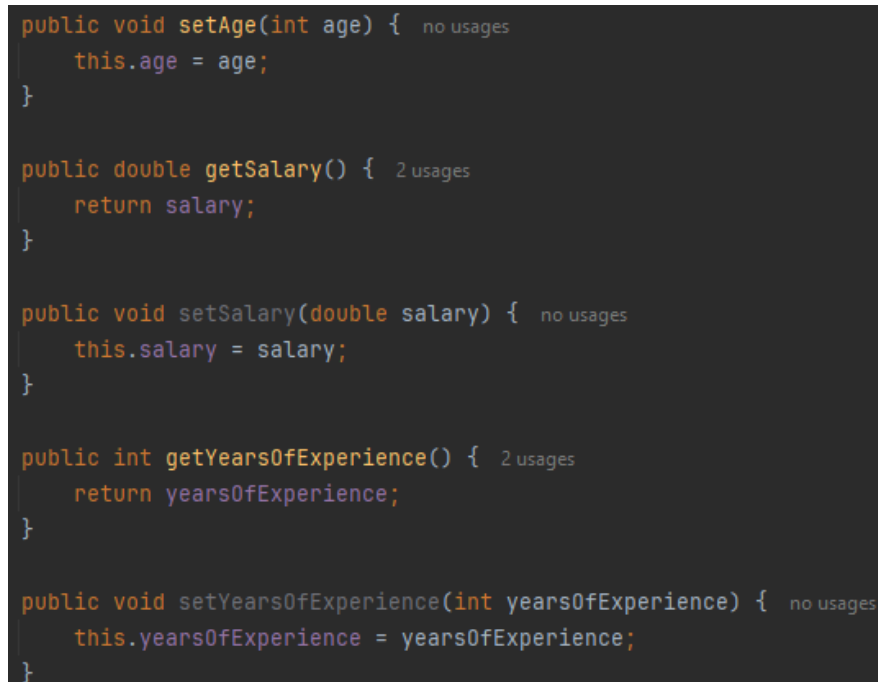
- Этот метод выводит информацию о программисте в консоль.

Класс администратора (имя, возраст, зарплата, опыт работы):



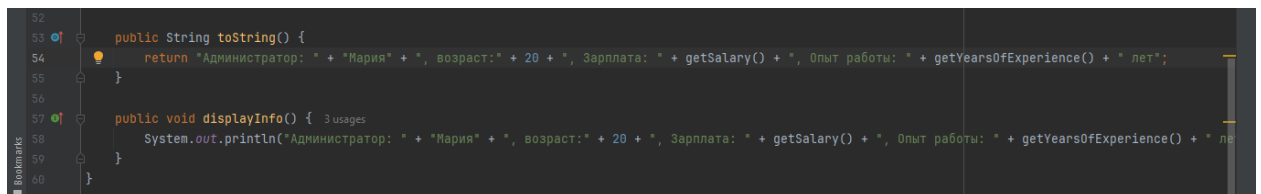
```
public class Administrator extends Employee {  
  
    private String name; // имя  
    private int age; //возраст  
    private double salary; // зарплата  
    private int yearsOfExperience; // опыт работы  
  
    public Administrator(String name, int age, double salary, int yearsOfExperience) {  
        super(name, age);  
        this.salary = salary;  
        this.yearsOfExperience = yearsOfExperience;  
    }  
  
    public Administrator() {  
        this("Мария", 35, 0, 0);  
    }  
  
    public String getName() {  
        return name;  
    }  
  
    public void setName(String name) {  
        this.name = name;  
    }  
  
    public int getAge() {  
        return age;  
    }  
}
```

Рисунок 6



```
    public void setAge(int age) {  
        this.age = age;  
    }  
  
    public double getSalary() {  
        return salary;  
    }  
  
    public void setSalary(double salary) {  
        this.salary = salary;  
    }  
  
    public int getYearsOfExperience() {  
        return yearsOfExperience;  
    }  
  
    public void setYearsOfExperience(int yearsOfExperience) {  
        this.yearsOfExperience = yearsOfExperience;  
    }  
}
```

Рисунок 7



```
52  
53 public String toString() {  
54     return "Администратор: " + "Мария" + ", возраст: " + 20 + ", Зарплата: " + getSalary() + ", Опыт работы: " + getYearsOfExperience() + " лет";  
55 }  
56  
57 public void displayInfo() {  
58     System.out.println("Администратор: " + "Мария" + ", возраст: " + 20 + ", Зарплата: " + getSalary() + ", Опыт работы: " + getYearsOfExperience() + " лет");  
59 }  
60 }
```

Рисунок 8

Был создан класс `Administrator`, который также наследуется от абстрактного класса `Employee`, и представляет администратора с уникальными атрибутами.

### 1. Наследование от `Employee`:

```
public class Administrator extends Employee {
```

- `public class Administrator`: определяет класс `Administrator`, который является подклассом `Employee`.
- `extends Employee`: указывает, что класс `Administrator` наследует все свойства и методы от класса `Employee`.

### 2. Поля:

```
private String name;
```

```
private int age;
```

```
private double salary;
```

```
private int yearsOfExperience;
```

- `private`: означает, что поля доступны только внутри класса `Administrator`.
- `name`: строка, хранящая имя администратора.
- `age`: целое число, хранящее возраст администратора.
- `salary`: вещественное число, хранящее зарплату администратора.
- `yearsOfExperience`: целое число, хранящее количество лет опыта работы администратора.

### 3. Конструкторы:

```
public Administrator(String name, int age, double salary, int yearsOfExperience) {
```

```
    super(name, age);
```

```
    this.salary = salary;
```

```
this.yearsOfExperience = yearsOfExperience;  
}
```

```
public Administrator() {  
  
    this("Мария", 29, 25000, 5);  
}
```

- **Первый конструктор:**

- public Administrator(String name, int age, double salary, int yearsOfExperience): инициализирует объект класса Administrator с указанными значениями для всех полей.
- super(name, age);: вызывает конструктор базового класса Employee, передавая ему имя и возраст.
- this.salary = salary;: присваивает значение поля salary параметру salary.
- this.yearsOfExperience = yearsOfExperience;: присваивает значение поля yearsOfExperience параметру yearsOfExperience.

- **Второй конструктор:**

- public Administrator(): инициализирует объект класса Administrator со значениями по умолчанию.
- this("Мария", 29, 25000, 5);: вызывает первый конструктор с указанными значениями по умолчанию.

#### 4. Геттеры и Сеттеры:

```
public String getName() { return name; }
```

```
public void setName(String name) { this.name = name; }
```

```
public int getAge() { return age; }
```

```
public void setAge(int age) { this.age = age; }
```

```
public double getSalary() { return salary; }
```

```
public void setSalary(double salary) { this.salary = salary; }
```

```
public int getYearsOfExperience() { return yearsOfExperience; }
```

```
public void setYearsOfExperience(int yearsOfExperience) {  
this.yearsOfExperience = yearsOfExperience; }
```

- Эти методы обеспечивают контролируемый доступ к значениям полей.

#### 5. Переопределенный метод toString():

```
@Override
```

```
public String toString() {
```

```
    return "Администратор: " + "Мария" + ", возраст:" + 29 + ", Зарплата: " +  
getSalary() + ", Опыт работы: " + getYearsOfExperience() + " лет";
```

```
}
```

- @Override: указывает, что этот метод переопределяет метод toString() из базового класса.
- Метод возвращает строку с информацией об администраторе.

#### 6. Реализация метода displayInfo():

```
public void displayInfo() {
```

```
    System.out.println("Администратор: " + "Мария" + ", возраст:" + 29 + ",  
Зарплата: " + getSalary() + ", Опыт работы: " + getYearsOfExperience() + "  
лет");
```

```
}
```

- Этот метод выводит информацию об администраторе в консоль.

Класс Manager(имя, возраст, зарплата, опыт работы):

```
Employee.java Main.java Programmist.java Administrator.java Manager.java
public class Manager extends Employee { 2 usages

    private String name; // имя 2 usages
    private int age; // возраст 2 usages
    private double salary; // зарплата 3 usages
    private int yearsOfExperience; // опыт работы (int вместо String) 3 usages

    public Manager(String name, int age, double salary, int yearsOfExperience) { 2 usages
        super(name, age);
        this.salary = salary;
        this.yearsOfExperience = yearsOfExperience;
    }

    public Manager() { no usages
        this( name: "известен", age: 20, salary: 25000, yearsOfExperience: 0); // Передайте значения в конструктор с параметрами
    }
```

Рисунок 9

```
public String getName() { no usages
    return name;
}

public void setName(String name) { no usages
    this.name = name;
}

public int getAge() { no usages
    return age;
}

public void setAge(int age) { no usages
    this.age = age;
}
```

Рисунок 10

```
public double getSalary() { 2 usages
    return salary;
}

public void setSalary(double salary) { no usages
    this.salary = salary;
}

public int getYearsOfExperience() { 2 usages
    return yearsOfExperience;
}

public void setYearsOfExperience(int yearsOfExperience) { no usages
    this.yearsOfExperience = yearsOfExperience;
}

@Override
public String toString() {
    return "Менеджер: " + "Илья" + ", возраст: " + 21 + ", Зарплата: " + getSalary() + ", Опыт работы: " + getYearsOfExperience() + " лет";
}

public void displayInfo() { 3 usages
    System.out.println("Менеджер: " + "Илья" + ", возраст: " + 21 + ", Зарплата: " + getSalary() + ", Опыт работы: " + getYearsOfExperience() + " лет");
}
```

Рисунок 11

Создали класс `Manager`, который также наследуется от абстрактного класса `Employee`, и представляет менеджера с уникальными атрибутами.

### 1. Наследование от `Employee`:

```
public class Manager extends Employee {
```

- `public class Manager`: определяет класс `Manager`, который является подклассом `Employee`.
- `extends Employee`: указывает, что класс `Manager` наследует все свойства и методы от класса `Employee`.

### 2. Поля:

```
private String name;
```

```
private int age;
```

```
private double salary;
```

```
private int yearsOfExperience;
```

- `private`: означает, что поля доступны только внутри класса `Manager`.
- `name`: строка, хранящая имя менеджера.
- `age`: целое число, хранящее возраст менеджера.
- `salary`: вещественное число, хранящее зарплату менеджера.
- `yearsOfExperience`: целое число, хранящее количество лет опыта работы менеджера.

### 3. Конструкторы:

```
public Manager(String name, int age, double salary, int yearsOfExperience) {
```

```
    super(name, age);
```

```
    this.salary = salary;
```

```
    this.yearsOfExperience = yearsOfExperience;
```

```
}
```

```
public Manager() {  
  
    this("Илья", 30, 25000, 6);  
  
}
```

- **Первый конструктор:**

- `public Manager(String name, int age, double salary, int yearsOfExperience)`: инициализирует объект класса `Manager` с указанными значениями для всех полей.
- `super(name, age);`: вызывает конструктор базового класса `Employee`, передавая ему имя и возраст.
- `this.salary = salary;`: присваивает значение поля `salary` параметру `salary`.
- `this.yearsOfExperience = yearsOfExperience;`: присваивает значение поля `yearsOfExperience` параметру `yearsOfExperience`.

- **Второй конструктор:**

- `public Manager()`: инициализирует объект класса `Manager` со значениями по умолчанию.
- `this("Илья", 30, 25000, 6);`: вызывает первый конструктор с указанными значениями по умолчанию.

#### 4. Геттеры и Сеттеры:

```
public String getName() { return name; }
```

```
public void setName(String name) { this.name = name; }
```

```
public int getAge() { return age; }
```

```
public void setAge(int age) { this.age = age; }
```

```
public double getSalary() { return salary; }
```

```
public void setSalary(double salary) { this.salary = salary; }
```

```
public int getYearsOfExperience() { return yearsOfExperience; }
```



```
public void setYearsOfExperience(int yearsOfExperience) {  
    this.yearsOfExperience = yearsOfExperience; }
```

- Эти методы обеспечивают контролируемый доступ к значениям полей.

**\*\*5. Переопределенный метод toString(): \*\***

@Override

```
public String toString() {  
  
    return "Менеджер: " + "Илья" + ", возраст:" + 30 + ", Зарплата: " +  
    getSalary() + ", Опыт работы: " + getYearsOfExperience() + " лет";  
  
}
```

- @Override: указывает, что этот метод переопределяет метод toString() из базового класса.
- Метод возвращает строку с информацией о менеджере.

**6. Реализация метода displayInfo():**

```
public void displayInfo() {  
  
    System.out.println("Менеджер: " + "Илья" + ", возраст:" + 30 + ", Зарплата: "  
    + getSalary() + ", Опыт работы: " + getYearsOfExperience() + " лет");  
  
}
```

- Этот метод выводит информацию о менеджере в консоль.