МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ

Ордена Трудового Красного Знамени

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский технический университет связи и информатики»

Кафедра «Математическая Кибернетика и Информационные технологии»

Лабораторная работа № 2

ООП

Выполнил: Студент группы

БВТ2303

Ситникова Дарья

Вариант 3

Москва

2024

Цель работы: изучение и практическое применение основных концепций объектно-ориентированного программирования (ООП) на языке Java, таких как абстракция, инкапсуляция, полиморфизм и наследование, через создание иерархии классов и реализацию различных методов и свойств объектов.

Задачи:

1. Создать иерархию классов, включающую:
    - Абстрактный класс.
    - Два уровня наследуемых классов, каждый из которых должен содержать минимум три поля и два метода, описывающих поведение объекта.
2. Демонстрировать реализацию всех принципов ООП, включая:
    - Абстракция.
    - Модификаторы доступа.
    - Перегрузка методов.
    - Переопределение методов.
3. Реализовать конструкторы для всех классов, включая конструкторы по умолчанию.
4. Реализовать геттеры и сеттеры для полей классов.
5. Обеспечить ввод/вывод информации о создаваемых объектах.
6. В одном из классов реализовать счетчик созданных объектов с использованием статической переменной и продемонстрировать его работу.

Краткая теория:

Объектно-ориентированное программирование (ООП) — это парадигма программирования, основанная на концепции "объектов", которые могут содержать данные и код: данные в виде полей (атрибутов), а код в виде методов (функций). ООП позволяет моделировать сложные системы, делая код более структурированным и удобным для понимания и сопровождения.

Основные концепции ООП

Абстракция — это процесс выделения общих характеристик объектов и игнорирование несущественных деталей. Это позволяет разработчикам сосредоточиться на том, что делает объект, а не на том, как он это делает. В Java абстракция может быть реализована с помощью классов и интерфейсов.

Инкапсуляция — это механизм, который ограничивает доступ к внутренним состояниям объекта и защищает его от некорректного использования. Это достигается с помощью модификаторов доступа (например, private, public).

Полиморфизм позволяет объектам разных классов обрабатывать данные по-разному, используя один и тот же интерфейс. Это означает, что вы можно использовать один и тот же метод для объектов разных классов, и каждый объект будет выполнять его по-своему. Полиморфизм обычно достигается через наследование и интерфейсы.

Наследование позволяет создавать новый класс на основе существующего, унаследовав его свойства и методы. Это помогает избежать дублирования кода и облегчает его поддержку.

Принципы ООП

- **Модификаторы доступа**:
    - private: доступен только внутри класса.
    - protected: доступен внутри класса и его подклассов, а также классам в одном пакете.
    - public: доступен из любого места.

- (по умолчанию) доступен только в пределах одного пакета.

- **Конструкторы**: Конструкторы — это специальные методы, которые вызываются при создании объекта. Они могут быть перегружены, и если суперкласс не имеет конструктора по умолчанию, подкласс должен явно вызывать конструктор суперкласса.
- **Статические переменные и методы**: Статические переменные принадлежат классу, а не конкретному объекту. Они могут использоваться для хранения информации, общей для всех экземпляров класса, например, счетчика созданных объектов.
- **Абстрактные классы и интерфейсы**: Абстрактные классы не могут быть инстанцированы и могут содержать как абстрактные методы (без реализации), так и обычные методы. Интерфейсы содержат только абстрактные методы и могут быть реализованы несколькими классами, что позволяет создавать гибкие и расширяемые системы.

Ход работы

В данной реализации создана иерархия классов, которая включает абстрактный класс Person и два уровня наследуемых классов: Student и Teacher, а также класс Assistant, который наследует от Teacher.

1. Абстрактный класс Person:
   - Поля

```
abstract class Person {
    private String name;
    private int age;
    private String gender;
```

   - Методы
     - Конструктор для инициализации полей

```
    public Person(String name, int age, String gender) {
        this.name = name;
        this.age = age;
        this.gender = gender;
    }
```

- Конструктор по умолчанию

```
public Person() {
    this("Неизвестно", 0, "Неизвестно");
}
```

- Геттеры и сеттеры для полей

```
//Геттер для имени
public String getName() {
    return name;
}

//Сеттер для имени
public void setName(String name) {
    this.name = name;
}

public int getAge() {
    return age;
}

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

public String getGender() {
    return gender;
}

public void setGender(String gender) {
    this.gender = gender;
}
```

- Абстрактный метод displayInfo(), который должен быть реализован в подклассах

```
public abstract void displayInfo();
}
```

2. Класс Student (наследуется от Person):
    - Поля:

```
class Student extends Person {
    private String major;
    private double gpa;
    private static int studentCount = 0;
```

- major (специальность)
- gpa (средний балл)
- studentCount (статическая переменная для подсчета созданных объектов)

- Методы:
  - Конструктор для инициализации полей и увеличения счетчика студентов.

```
public Student(String name, int age, String gender, String major, double gpa) {
    super(name, age, gender);
    this.major = major;
    this.gpa = gpa;
    studentCount++;
}
```

  - Конструктор по умолчанию.

```
public Student() {
    this("Неизвестно", 0, "Неизвестно", "Неизвестно", 0.0);
}
```

  - Геттеры и сеттеры для полей.

```
public String getMajor() {
    return major;
}

public void setMajor(String major) {
    this.major = major;
}

public double getGpa() {
    return gpa;
}

public void setGpa(double gpa) {
    this.gpa = gpa;
}
```

  - Переопределенный метод displayInfo(), который выводит информацию о студенте.

```
@Override
public void displayInfo() {
    System.out.println("Студент: " + getName() + ", Возраст: " + getAge() + ", Пол: " + getGender() +
            ", Специальность: " + major + ", GPA: " + gpa);
}
```

- Статический метод getStudentCount(), возвращающий количество созданных студентов.

```
    public static int getStudentCount() {
        return studentCount;
    }
}
```

3. Класс Teacher (наследуется от Person):
   - Поля:

```
class Teacher extends Person {
    private String subject;
    private int experience;
```

     - subject (предмет)
     - experience (опыт работы)
   - Методы:
     - Конструктор для инициализации полей.

```
public Teacher(String name, int age, String gender, String subject, int experience) {
    super(name, age, gender);
    this.subject = subject;
    this.experience = experience;
}
```

     - Конструктор по умолчанию.

```
public Teacher() {
    this("Неизвестно", 0, "Неизвестно", "Неизвестно", 0);
}
```

     - Геттеры и сеттеры для полей.

```java
public String getSubject() {
    return subject;
}

public void setSubject(String subject) {
    this.subject = subject;
}

public int getExperience() {
    return experience;
}

public void setExperience(int experience) {
    this.experience = experience;
}
```

- Переопределенный метод displayInfo(), который выводит информацию о преподавателе.

```java
@Override
public void displayInfo() {
    System.out.println("Преподаватель: " + getName() + ", Возраст: " + getAge() + ", Пол: " + getGender() +
            ", Предмет: " + subject + ", Опыт: " + experience + " лет");
}
}
```

4. Класс Assistant (наследуется от Teacher):
   - Поля:

```java
class Assistant extends  Teacher {
    private String officeHours;
```

     - officeHours (часы работы)
   - Методы:
     - Конструктор для инициализации полей, включая поля суперкласса.

```java
public Assistant(String name, int age, String gender, String subject, int experience, String officeHours) {
    super(name, age, gender, subject, experience);
    this.officeHours = officeHours;
}
```

     - Конструктор по умолчанию.

```java
public Assistant() {
    this("Неизвестно", 0, "Неизвестно", "Неизвестно", 0, "Неизвестно");
}
```

     - Геттер и сеттер для поля officeHours.

```
    public String getOfficeHours() {
        return officeHours;
    }

    public void setOfficeHours(String officeHours) {
        this.officeHours = officeHours;
    }
```

- Переопределенный метод displayInfo(), который выводит информацию об ассистенте.

```
    @Override
    public void displayInfo() {
        System.out.println("Ассистент: " + getName() + ", Возраст: " + getAge() + ", Пол: " + getGender() +
                ", Предмет: " + getSubject() + ", Опыт: " + getExperience() + " лет, Часы работы: " + officeHours
    }
}
```

В классе Main создаются экземпляры классов Student, Teacher и Assistant. Для каждого объекта вызывается метод displayInfo(), который выводит информацию о созданных объектах. В конце программы выводится общее количество созданных студентов с помощью статического метода getStudentCount().

```
public class Main {
    public static void main(String[] args) {
        Student student1 = new Student("Натлья", 19, "Женский", "Информатика", 4.8);
        Student student2 = new Student("Илья", 21, "Мужской", "Информатика", 4.5);
        Teacher teacher1 = new Teacher("Сергей", 45, "Мужской", "Математика", 20);
        Assistant assistant1 = new Assistant("Елена", 30, "Женский", "Химия", 5, "Пн-Пт 10:00-12:00");


        student1.displayInfo();
        student2.displayInfo();
        teacher1.displayInfo();
        assistant1.displayInfo();

        System.out.println("Количество созданных студентов: " + Student.getStudentCount());
    }
}
```

Результат:

```
PS C:\Users\manul\Desktop\ИТиП> java Main.java
Студент: Натлья, Возраст: 19, Пол: Женский, Специальность: Информатика, GPA: 4.8
Студент: Илья, Возраст: 21, Пол: Мужской, Специальность: Информатика, GPA: 4.5
Преподаватель: Сергей, Возраст: 45, Пол: Мужской, Предмет: Математика, Опыт: 20 лет
Ассистент: Елена, Возраст: 30, Пол: Женский, Предмет: Химия, Опыт: 5 лет, Часы работы: Пн-Пт 10:00-12:00
Количество созданных студентов: 2
PS C:\Users\manul\Desktop\ИТиП>
```

Реализация принципов ООП

1. Абстракция:

- Использование абстрактного класса Person позволяет скрыть детали реализации и предоставляет общий интерфейс для всех классов, наследуемых от него.

2. Инкапсуляция:
   - Поля классов имеют модификатор доступа private, что защищает их от прямого доступа извне. Доступ к полям осуществляется через геттеры и сеттеры.
3. Наследование:
   - Классы Student, Teacher и Assistant наследуют от абстрактного класса Person, что позволяет им использовать его методы и поля, а также добавлять свои собственные.
4. Полиморфизм:
   - Переопределение метода displayInfo() в подклассах позволяет каждому классу предоставлять свою реализацию этого метода, что демонстрирует полиморфизм.
5. Конструкторы:
   - Каждый класс имеет как конструктор для инициализации полей, так и конструктор по умолчанию, что позволяет создавать объекты с заданными значениями или с значениями по умолчанию.
6. Статистические переменные:
   - Каждый класс имеет как конструктор для инициализации полей, так и конструктор по умолчанию, что позволяет создавать объекты с заданными значениями или с значениями по умолчанию.

Вывод: В ходе выполнения лабораторной работы была успешно разработана иерархия классов, демонстрирующая основные принципы объектно-ориентированного программирования (ООП) на языке Java.

Приложение:

GitHub: https://github.com/Daria0w0/ITiP

1. **Person.java**

```java
abstract class Person {
    private String name;
    private int age;
    private String gender;

    // Конструктор, инициализирующий поля класса
    public Person(String name, int age, String gender) {
        this.name = name;
        this.age = age;
        this.gender = gender;
    }

    //Конструктор по умолчанию
    public Person() {
        this("Неизвестно", 0, "Неизвестно");
    }

    //Геттер для имени
    public String getName() {
        return name;
    }

    //Сеттер для имени
    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

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

    public String getGender() {
        return gender;
    }

    public void setGender(String gender) {
        this.gender = gender;
    }

    // Абстрактный метод для вывода информации о человеке
    public abstract void displayInfo();
}
```

2. Student.java

```
class Student extends Person {
    private String major;
    private double gpa;
    private static int studentCount = 0;

    public Student(String name, int age, String gender, String major, double gpa)
{
        super(name, age, gender);
        this.major = major;
        this.gpa = gpa;
        studentCount++;
    }

    public Student() {
        this("Неизвестно", 0, "Неизвестно", "Неизвестно", 0.0);
    }

    public String getMajor() {
        return major;
    }

    public void setMajor(String major) {
        this.major = major;
    }

    public double getGpa() {
        return gpa;
    }

    public void setGpa(double gpa) {
        this.gpa = gpa;
    }

    @Override
    public void displayInfo() {
        System.out.println("Студент: " + getName() + ", Возраст: " + getAge() +
", Пол: " + getGender() +
                ", Специальность: " + major + ", GPA: " + gpa);
    }

    public static int getStudentCount() {
        return studentCount;
    }

}
```

3. Teacher.java

```
class Teacher extends Person {
```

```
    private String subject;
    private int experience;

    public Teacher(String name, int age, String gender, String subject, int
experience) {
        super(name, age, gender);
        this.subject = subject;
        this.experience = experience;
    }

    public Teacher() {
        this("Неизвестно", 0, "Неизвестно", "Неизвестно", 0);
    }

    public String getSubject() {
        return subject;
    }

    public void setSubject(String subject) {
        this.subject = subject;
    }

    public int getExperience() {
        return experience;
    }

    public void setExperience(int experience) {
        this.experience = experience;
    }

    @Override
    public void displayInfo() {
        System.out.println("Преподаватель: " + getName() + ", Возраст: " +
getAge() + ", Пол: " + getGender() +
                ", Предмет: " + subject + ", Опыт: " + experience + " лет");
    }
}
```

4. Assistant.java

```
class Assistant extends  Teacher {
    private String officeHours;

    public Assistant(String name, int age, String gender, String subject, int
experience, String officeHours) {
        super(name, age, gender, subject, experience);
        this.officeHours = officeHours;
    }

    public Assistant() {
```

```
        this("Неизвестно", 0, "Неизвестно", "Неизвестно", 0, "Неизвестно");
    }

    public String getOfficeHours() {
        return officeHours;
    }

    public void setOfficeHours(String officeHours) {
        this.officeHours = officeHours;
    }

    @Override
    public void displayInfo() {
        System.out.println("Ассистент: " + getName() + ", Возраст: " + getAge() +
", Пол: " + getGender() +
                ", Предмет: " + getSubject() + ", Опыт: " + getExperience() + "
лет, Часы работы: " + officeHours);
    }
}
```

5. Main.java

```
public class Main {
    public static void main(String[] args) {
        Student student1 = new Student("Натлья", 19, "Женский", "Информатика",
4.8);
        Student student2 = new Student("Илья", 21, "Мужской", "Информатика",
4.5);
        Teacher teacher1 = new Teacher("Сергей", 45, "Мужской", "Математика",
20);
        Assistant assistant1 = new Assistant("Елена", 30, "Женский", "Химия", 5,
"Пн-Пт 10:00-12:00");


        student1.displayInfo();
        student2.displayInfo();
        teacher1.displayInfo();
        assistant1.displayInfo();

        System.out.println("Количество созданных студентов: " +
Student.getStudentCount());
    }
}
```