Лабораторная работа 3

Цель работы: изучить реализацию принципов объектно-ориентированного программирования в языке Java.

Базовым элементом объектно-ориентированного программирования в языке Java является класс.

Для объявления класса служит ключевое слово **class**:

```
public class MainActivity extends Activity {
    // код внутри класса
}
```

Упрощённая общая форма для класса может иметь следующий вид:

```
class ИмяКласса {
    тип переменнаяЭкземпляра;

    тип имяМетода(список параметров){
        // тело метода
    }
}
```

В Java принято начинать имена класса с большой буквы. В классе могут быть несколько переменных и методов. Переменные, определённые внутри класса (не метода), называются переменными экземпляра или полями (*fields*). Код пишется внутри класса. Методы и переменные внутри класса являются членами класса.

Имя исходного файла Java должно соответствовать имени хранящегося в нем класса. Регистр букв важен и в имени класса, и в имени файла.

Класс — это шаблон для создания объекта. Класс определяет структуру объекта и его методы, образующие функциональный интерфейс. В процессе выполнения Java-программы система использует определения классов для создания представителей классов. Представители являются реальными объектами. Термины «*представитель*», «*экземпляр*» и «*объект*» взаимозаменяемы.

```
class Student{

    String name;          // имя
    int age;              // возраст
    void displayInfo(){
        System.out.printf("Name: %s \tAge: %d\n",
                          name, age);
    }
}
```

В классе Student определены два поля: *name* представляет имя человека, а *age* - его возраст. И также определен метод *displayInfo*, который ничего не возвращает и просто выводит эти данные на консоль.

Новый объект (или экземпляр) создаётся из существующего класса при помощи ключевого слова **new**.

Как правило, классы определяются в разных файлах. В данном случае для простоты определяем два класса в одном файле. Стоит отметить, что в этом случае только один класс может иметь модификатор **public** (в данном случае это класс *Program*), а сам файл кода должен называться по имени этого класса, то есть в данном случае файл должен называться *Program.java*.

Класс представляет новый *mun*, поэтому мы можем определять переменные, которые представляют данный тип. Так, здесь в методе *main* определена переменная *misha*, которая представляет класс *Student*. Но пока эта переменная не указывает ни на какой объект и по умолчанию она имеет значение *null*. По большому счету мы ее пока не можем использовать, поэтому вначале необходимо создать объект класса *Student*.

```
public class Program{

    public static void main(String[] args) {

        Student misha;
    }
}
class Student{
    String name;    // имя
    int age;        // возраст
    void displayInfo(){
        System.out.printf("Name: %s \tAge: %d\n", name,
age);
    }
}
```

Конструкторы

Кроме обычных методов классы могут определять специальные методы, которые называются конструкторами. Конструкторы вызываются при создании нового объекта данного класса. Конструкторы выполняют инициализацию объекта.

Если в классе не определено ни одного конструктора, то для этого класса автоматически создается конструктор без параметров.

Выше определенный класс *Student* не имеет никаких конструкторов. Поэтому для него автоматически создается конструктор по умолчанию, который мы можем использовать для создания объекта *Student*. В частности, создадим один объект:

```
public class Program{

    public static void main(String[] args) {

        Student misha = new Student(); // создание
объекта
        misha.displayInfo();

        // изменяем имя и возраст
        misha.name = "Михаил";
        misha.age = 20;
        misha.displayInfo();
    }
}
class Student{
    String name;    // имя
    int age;        // возраст
    void displayInfo(){
       System.out.printf("Name: %s \tAge: %d\n", name,
age);
    }
}
```

Для создания объекта *Student* используется выражение **new** *Student()*. Оператор **new** выделяет память для объекта *Student*. И затем вызывается конструктор по умолчанию, который не принимает никаких параметров. В итоге после выполнения данного выражения в памяти будет выделен участок, где будут храниться все данные объекта *Student*. А переменная *misha* получит ссылку на созданный объект.

Если конструктор не инициализирует значения переменных объекта, то они получают значения по умолчанию. Для переменных числовых типов это число 0, а для типа *string* и классов - это значение *null* (то есть фактически отсутствие значения).

После создания объекта мы можем обратиться к переменным объекта *Student* через переменную *misha* и установить или получить их значения, например, *misha.name = "Михаил"*.

Если необходимо, что при создании объекта производилась какая-то логика, например, чтобы поля класса получали какие-то определенные значения, то можно определить в классе свои конструкторы. Например:

```
public class Program{

    public static void main(String[] args) {

        Student dima = new Student();        // вызов
первого конструктора без параметров
```

```
        dima.displayInfo();

        Student misha = new Student("Михаил"); // вызов
второго конструктора с одним параметром
        misha.displayInfo();

        Student sasha = new Student("Александр", 21);
// вызов третьего конструктора с двумя параметрами
        sasha.displayInfo();
    }
}

class Student{
    String name;    // имя
    int age;        // возраст
    Student() {
        name = "Undefined";
        age = 18;
    }
    Student(String n) {
        name = n;
        age = 18;
    }
    Student(String n, int a) {
        name = n;
        age = a;
    }
    void displayInfo(){
        System.out.printf("Name: %s \tAge: %d\n", name,
age);
    }
}
```

Ключевое слово this

Ключевое слово *this* представляет ссылку на текущий экземпляр класса. Через это ключевое слово мы можем обращаться к переменным, методам объекта, а также вызывать его конструкторы. Например:

```
public class Program{

    public static void main(String[] args) {

        Student undef = new Student();
        undef.displayInfo();
```

```
        Student misha = new Student("Михаил");
        misha.displayInfo();

        Student dima = new Student("Дмитрий", 23);
        dima.displayInfo();
    }
}
class Student{
    String name;    // имя
    int age;        // возраст
    Student() {
        this("Undefined", 18);
    }
    Student(String name) {
        this(name, 18);
    }
    Student(String name, int age) {
        this.name = name;
        this.age = age;
    }
    void displayInfo() {
        System.out.printf("Name: %s \tAge: %d\n", name,
age);
    }
}
```

В третьем конструкторе параметры называются так же, как и поля класса. И чтобы разграничить поля и параметры, применяется ключевое слово **this**:

```
this.name = name;
```

Так, в данном случае указываем, что значение параметра *name* присваивается полю *name*.

Кроме того, у нас три конструктора, которые выполняют идентичные действия: устанавливают поля *name* и *age*. Чтобы избежать повторов, с помощью **this** можно вызвать один из конструкторов класса и передать для его параметров необходимые значения:

```
Student(String name) {
    this(name, 18);
}
```

В итоге результат программы будет тот же, что и в предыдущем примере.

Доступ к полям класса

Следующее понятие из мира ООП, которое следует рассмотреть - это геттеры и сеттеры (getter - от англ. "get" - получать, и setter - от англ.

"set" - устанавливать). Это общепринятый способ вводить данные ("set") или получать данные ("get").

Названия методов чаще всего состоят из слова get/set + названия поля, за которое они отвечают.

```
public class Student {

    private String name;
    private int age;

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

Метод `getName` (имя метода-геттера состоит из слова get + название переменной) - это метод, который возвращает переменную типа `String`, и при этом не требует никаких данных.

Метод `setName` (имя метода-сеттера тоже состоит из слова set + название переменной) ничего не возвращает (void) но при этом требует String - новое значение для переменной name.

В ООП геттеры и сеттеры реализуют принцип инкапсуляции С помощью геттеров и сеттеров программист защищает содержимое программы - когда ей пользуется кто-то другой.

Принцип работы состоит в выполнении следующих этапов:

1. Задать полям, которые мы хотим защитить, свойство `private`.
2. Написать метод геттер/ сеттер для каждого поля.

Основные правила:

1. Наименование полей (переменных, содержащихся в классе) всегда пишется с маленькой буквы (например, `int number`, `String name`, и т.д.).

2. Как уже говорилось, наименование геттеров и сеттеров - в формате "`get`" + имя переменной с большой буквы (например, `getColor`, `getName`).

3. Метод *геттер* не имеет параметров (т.е. в скобках ничего не пишется) и возвращает значение одной переменной (одного поля).

4. Метод *сеттер* всегда имеет модификатор `void` и только один параметр, для изменения значения одного поля.

Задания на лабораторную работу:

Написать программу, в которой описать классы по заданной тематике. В каждом классе должно быть не менее пяти различных полей, трех конструкторов и трех методов. Все поля должны иметь модификатор `private`. Для доступа к этим полям реализовать геттеры и сеттеры. Реализовать переопределенные методы `toString()` и `equals()`.

Создать объекты этих классов, занести информацию в поля этих классов.

Вариант 1	*Геометрические фигуры*	Вариант 7	*Учет аудиторного фонда*
Вариант 2	*Библиотека*	Вариант 8	*Учет оборудования*
Вариант 3	*Магазин продуктов*	Вариант 9	*Экзаменационная комиссия*
Вариант 4	*Магазин промтоваров*	Вариант 10	*Дистанционное управление техникой*
Вариант 5	*Отдел кадров*	Вариант 11	*Планировка помещений*
Вариант 6	*Автомастерская*	Вариант 12	*Интернет-провайдер*

Содержание отчета по лабораторной работе

- Титульный лист
- Описание задания
- Код разработанного класса
- Пример использования разработанного класса

Вопросы для самоконтроля

1. Какие принципы составляют понятие объектно-ориентированного программирования?
2. Как в языке Java реализуется принцип инкапсуляции данных?
3. Как реализуется наследование и какие имеет ограничения?
4. Что такое геттеры и сеттеры?

Список литературы

1. Вязовик, Н. А. Программирование на Java : учебное пособие / Н. А. Вязовик. — 3-е изд. — Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2021. — 601 c. — ISBN 978-5-4497-0852-6. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/102048.html
2. Мухаметзянов, Р. Р. Основы программирования на Java : учебное пособие / Р. Р. Мухаметзянов. — Набережные Челны : Набережночелнинский государственный педагогический университет, 2017. — 114 c. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/66812.html
3. Блох, Дж. Java. Эффективное программирование / Дж. Блох ; перевод В. Стрельцов ; под редакцией Р. Усманов. — 2-е изд. — Саратов : Профобразование, 2019. — 310 c. — ISBN 978-5-4488-0127-3. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/89870.html