

Частное учреждение образования
«Колледж бизнеса и права»

УТВЕРЖДАЮ

Ведущий

методист колледжа

_____Е.В. Паскал

«___»_____2021

Специальность: 2-40 01 01 «Программное обеспечение информационных технологий»	Учебная дисциплина: «Основы кроссплатформенного программирования»
---	---

Лабораторная работа № 2
Инструкционно-технологическая карта

Тема: «Разработка классов на языке Java»

Цель: Научиться создавать классы, методы и экземпляры классов, а также обращаться к ним в языке Java

Время выполнения: 4 часа

ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Изучить теоретические сведения;
2. Ответить на контрольные вопросы;
3. Откомпилировать примеры программ из раздела «Теоретические сведения»;
4. Выполнить ИДЗ.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ ДЛЯ ВЫПОЛНЕНИЯ РАБОТЫ

В этом лабораторной работе будет рассмотрено создание класса на языке java и последующее обращение к этому классу, то есть мы научимся основам работы с классами.

Создание класса в java

Создавать класс мы будем в программе Eclipse, но также можно работать в любой другой IDE или в блокноте. В качестве примера мы будем создавать класс, который описывает характеристики и поведение кота.

Создадим в нашем проекте новый класс под именем Cat. Для этого в меню Eclipse выберем File -> new -> class и, в открывшемся окне, введем имя класса – Cat. Обратите внимание, что галочка public static void main (String[] args) должна быть снята! Затем нажмем Finish

В итоге, у нас создался новый класс.

Структура класса в java

Класс в java оформляется с помощью следующей конструкции:

```
public class Cat { }
```

Рассмотрим ее составляющие:

`public` – модификатор доступа к классу, в данном случае он нам говорит, что этот класс будет доступен не только данному классу, но и другим. В java существуют и другие модификаторы, но об этом поговорим позднее.

`class` – ключевое слово, говорящее о том, что это класс.

`Cat` – имя класса. Имена классов принято писать с заглавной буквы.

`{ }` – фигурные скобки, между которыми разместится тело нашего класса.

В предыдущей лабораторной работе мы упоминали, что классы в java состоят из атрибутов и методов, присвоим некоторые и нашему коту.

Атрибуты класса Cat

Атрибутами кота могут быть: имя, вес, окраска. Атрибуты это переменные, которые объявляются следующим образом:

```
private int weight; // вес кота
private String name; // имя кота
private String color; //окрас кота
```

`private` –здесь опять же указывает на права доступа. К переменной, в данном случае, можно обращаться только из ее класса, чтобы классы извне не могли ее изменить.

`int`, `String` – это типы данных. В данном случае, вес будет задан при помощи целого числа – `int`, а имя и цвет при помощи символьной строки `String`.

После объявления каждого атрибута должна ставиться точка с запятой;

При помощи `//` в коде программы оформляются комментарии, которые являются правилом хорошего тона.

Методы класса Cat

Пусть наш кот умеет есть, спать и разговаривать. Опишем это поведение с помощью методов.

```
//кот ест public void eat(){ System.out.print("Eating...\n"); }
```

`public void eat()` – сигнатура метода, его определение. В данном случае она нам говорит о том, что:

- метод `eat()` доступен для других классов – `public`;
- метод `eat()` не возвращает никаких данных – `void`.

`eat` – имя метода. Имена методов, в отличие от имен классов, принято писать с маленькой буквы. На конце каждого метода после имени всегда идут круглые скобки `()`, пустые или содержащие параметры (об этом позднее).

После сигнатуры метода идут фигурные скобки `{ }`. В них содержится тело метода. Тело нашего метода содержит просто вывод сообщения о том, что кот кушает — `"Eating...\n"`. За вывод текстового сообщения отвечает метод `System.out.print("Eating...\n")`.

\n — символ перевода строки, чтобы при выводе наших сообщений все они не писались в одну строку.

Аналогично предыдущему методу опишем спящего кота:

```
//кот спит
```

```
public void sleep(){
System.out.print("Sleeping zz-z-z-z...\n"); }
```

И немного иначе опишем метод – кот говорит:

```
//кот говорит
```

```
public String speak(String words){
String phrase = words + "...mauu...\n"; return phrase; }
```

В отличие от предыдущих методов, этот метод возвращает значение и имеет входные параметры. Давайте подробнее рассмотрим сигнатуру метода `public String speak(String words)`:

`public`- метод `speak()` доступен для других классов;

`String` -тип значения, которое возвращает метод. В предыдущих случаях ключевое слово `void` указывало на то, что метод ничего не возвращает. В данном случае `String` указывает на то, что метод возвращает значение типа строка.

В процессе своей работы, метод выполняет определенные действия над данными, иногда необходимо, чтобы результат этих действий был передан для дальнейшей обработки другим классам, в этом случае метод передает (возвращает) этот результат. Эти возвращаемые данные, относятся к какому-либо типу, в нашем примере это тип символьной строки - `String`.

Возвращающие методы должны содержать в своем теле ключевое слово `return`, которое указывает на то, что именно возвращает данный метод. В нашем случае это переменная `phrase`.

`speak`- имя метода.

`(String words)` — входные параметры. Входные параметры, это какие-либо данные, которые передаются из других классов и, которые метод должен обработать. Наш метод получает в качестве входных данных строку в виде переменной `words`, к этой строке дописывает «...mauu...» и возвращает то, что получилось.

В итоге класс `Cat` выглядит следующим образом:

```
public class Cat {
private int weight; // вес кота
private String name; // имя кота
private String color; //окрас кота
//кот ест
public void eat(){
System.out.print("Eating...\n"); }
//кот спит
public void sleep(){
System.out.print("Sleeping zz-z-z-z...\n"); }
//кот говорит
public String speak(String words){
String phrase = words + "...mauu...\n"; return phrase; } }
```

Обращение к классу в Java

Далее рассмотрим обращение к классу Cat. И то, как его использовать в работе.

Создадим класс Hello: File -> new -> class, ввести в открывшемся окне имя класса и, в этот раз, обязательно установить галочку public static void main(String[] args).

public static void main(String[] args) – сигнатура метода main(). Программа состоит из нескольких классов, но только один из классов может содержать метод main(). Метод main(), это так называемая точка входа в программу. Без этого метода мы не сможем стартовать программу. Теперь, вместо приветствия миру, этот метод будет содержать код для работы с классом Cat.

Пишем в тело метода main() (между его фигурными скобками) следующие строки:

```
Cat ourcat = new Cat();
ourcat.eat();
ourcat.sleep();
String say = ourcat.speak("Play with me");
System.out.println(say);
```

Разберем подробнее, что это значит.

Прежде чем вызывать, созданные нами методы в классе Cat, и заставить нашего кота есть, спать и говорить, сперва нужно создать экземпляр класса(инстанцию).

```
Cat ourcat = new Cat();
```

Данная строчка нам говорит о том, что в памяти создан экземпляр объекта Cat, а переменная ourcat типа Cat (такого же, как и наш объект) указывает на то место в памяти, где был этот объект создан.

Переменную ourcat теперь можно использовать для вызова методов класса Cat, например:

```
ourcat.eat(); ourcat.sleep();
```

При вызове этих методов в программе Eclipse удобно пользоваться комбинацией клавиш

Ctrl + пробел, после введения имени переменной и точки. Программа подскажет, какие можно использовать методы для данной переменной.

Если метод возвращает какое-либо значение, например, как наш метод speak() возвращает значение типа String, то его можно вызывать следующим образом:

- объявить переменную такого же типа, что и возвращаемое значение (в нашем случае String)
- присвоить ей вызванный метод, например:

```
String say = ourcat.speak("Play with me");
```

Вспомним, что при описании нашего метода он содержал параметры speak(String words). Теперь, при вызове в качестве параметра выступила фраза "Play with me", метод speak() ее обработал и вернул "Play with me...maui...". Именно это значение он присвоил переменной say.

Мы это можем проверить, выведя say на печать при помощи команды:

```
System.out.println(say);
```

Итак, наш класс Hello теперь выглядит следующим образом:

```
public class Hello {
    /** * @param args */
    public static void main(String[] args) {
        Cat ourcat= new Cat();
        ourcat.eat();
        ourcat.sleep();
        String say = ourcat.speak("Play with me");
        System.out.println(say); } }
```

Теперь сохраним и запустим нашу программу. При запуске Eclipse может предложить выбрать Java Application. Нужно выбрать Java Application.

В качестве результата внизу в консоле мы получаем следующие строки.

Eating...Sleeping zz-z-z-z...Play with me...mauu...

В программировании очень важна безопасность. В ООП безопасность обеспечивается по-своему - с помощью принципа инкапсуляции (с англ. "encapsulation"). Инкапсуляцию можно перевести как «положить что-то во что-то», или для простоты "обернуть в капсулу"

С помощью инкапсуляции мы защищаем данные от неправомерного использования.

Точно так же и в Java - мы пользуемся разными средствами для обеспечения принципа инкапсуляции. Но как же мы это делаем?

Есть несколько способов регулировать доступ к нашим данным. Основные это:

- Модификаторы доступа (Access modifiers)
- Геттеры и Сеттеры (Getters and Setters)

Модификаторы доступа

Модификаторы доступа — это специальные слова, которые показывают, кому нельзя, а кому можно пользоваться данными.

Существуют четыре модификатора доступа:

- `public` - "публичный, доступный всем"
- `default` - "по умолчанию". Когда мы не пишем модификатора доступа (как мы это делали в наших предыдущих уроках), он по умолчанию имеет значение `default`. Данные с этим модификатором видны в пределах `package`.
 - `protected` - "защищенный". На самом деле это то же самое, что и `default`, только доступ имеют еще и классы-наследники.
 - `private` - "частный, личный". Такие данные видны только самому классу.

Модификаторы доступа пишутся перед названиями переменных, методов и даже классов:

Как это обеспечивает безопасность? Давайте попробуем создать класс, в котором будет только одна переменная - `String s`. Допустим она имеет модификатор `public`:

```
class MyClass {
    public String s = "Hello World!";
}
```

Теперь попробуем вывести значение этой переменной на экран:

```
public class Test {
    public static void main(String[] args) {
        MyClass obj = new MyClass();
        System.out.println(obj.s);
    }
}
```

Получили:

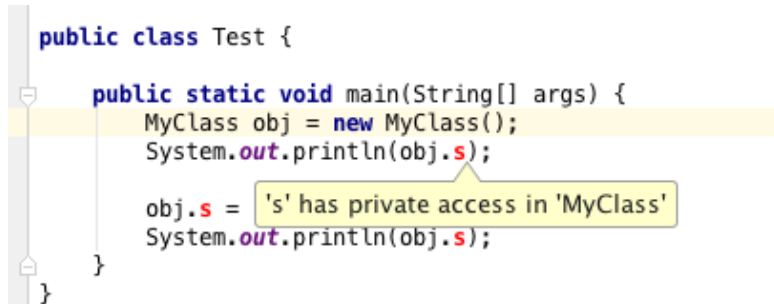
```
Hello World!
```

```
Process finished with exit code 0
```

Тем не менее, точно так же мы можем и поменять эту переменную. Если мы не хотим этого - поменяем модификатор с public на private:

```
class MyClass {
    private String s = "Hello World!";
}
```

Теперь при попытке доступа к переменной s - будь то чтение или запись - у нас возникнет ошибка:



```
public class Test {
    public static void main(String[] args) {
        MyClass obj = new MyClass();
        System.out.println(obj.s);

        obj.s = 's' has private access in 'MyClass'
        System.out.println(obj.s);
    }
}
```

Вот так мы защитили переменную.

Геттеры и Сеттеры

Ну, вот мы защитили переменную - но ее же надо как-то менять? Если влиять на переменную напрямую считается плохой практикой, то, как по-другому, правильно можно изменять переменные?

Для этого существуют специальные методы - так называемые Геттеры и Сеттеры. Ну, они не то чтобы специальные - просто настолько часто используются, что были вынесены в отдельную категорию методов.

Геттер - от англ. "get", "получать" — это метод, с помощью которого мы получаем значение переменной, т.е. ее читаем. Например, создадим Геттер для нашей переменной s:

Сеттер - от англ. "set", "устанавливать" — это метод, с помощью которого мы меняем, или задаем значение переменной. Допишем Сеттер для переменной s:

```
class MyClass {
    private String s = "Hello World!";

    public String getString()
    {
        return s;
    }
    public void setS(String newValue)
    {
        s = newValue;
    }
}
```

Теперь, если переписать наш main() по-правильному, получим:

```
public class Test {

    public static void main(String[] args) {
        MyClass obj = new MyClass();
        System.out.println(obj.getS());

        obj.setS("It's modified!");
        System.out.println(obj.getS());
    }
}
```

Как видите, вместо того чтобы напрямую стучаться к переменной, мы меняем ее с помощью геттеров и сеттеров. Мы получаем:

```
Hello World!
It's modified!

Process finished with exit code 0
```

Если результат такой же, зачем это все было менять?

Тут у нас в каждом методе всего по одной строчке, но если нам понадобится добавить какую-то логику - например, присваивать новое значение строке s только если она больше какой-то длины, или если содержит слово "Java". Главное, мы получаем контроль над происходящим - никто не может просто так менять или читать наши переменные.

Доступ к элементам класса, которые объявлены с модификатором доступа private.

Элементы класса, которые объявленные с модификатором доступа private скрыты. К ним имеют доступ только методы данного класса. Из всех других частей программного кода к private-членам класса нет доступа.

Такой способ скрытия данных в классе эффективный в случаях, если нужно:

- спрятать детали организации структур данных и реализации данного класса;
- избежать возможные случайные изменения данных в экземпляре класса;
- обеспечить удобство доступа к данным в классе с помощью специально разработанных методов. Здесь имеется ввиду разработка методов в классе которые обеспечивают чтение/запись данных класса. Эти методы могут включать соответствующие проверки на корректность при изменении данных в классе;
- избежать возможные злоупотребления данными в классе что может привести к возникновению трудноуловимых ошибок в программе.

Пример.

Объявляется класс, реализующий день недели. В классе объявляется скрытый (private) член данных класса с именем value. Для доступа к value в классе используются:

- конструктор DayWeek(), который инициализирует значение value=1. Конструктор объявлен без модификатора. Это значит, что в пределах пакета он имеет тип доступа public;
- метод Get(), возвращающий значение value. Поскольку метод объявлен в классе DayWeek, то в теле метода есть доступ к переменной value;
- метод Set(), устанавливающий новое значение value. Метод также объявлен в классе, поэтому имеет доступ к value. В методе осуществляется проверка на корректность значения value в границах 1..7. Если задать другое значение, то значение value не изменится;
- метод GetName(), возвращающий название дня недели в зависимости от значения value.

Реализация класса DayWeek следующая

// класс, который реализует день недели

```
class DayWeek {
    private int value; // скрытая переменная класса

    // конструктор класса, инициализирует переменную value
    DayWeek() {
        value = 1; // по умолчанию - "Monday"
    }
    // методы доступа
    // метод, который возвращает данные в классе
    public int Get() {
        return value;
    }
    // метод, который устанавливает новый день недели
    public void Set(int _value) {
        // в методе выполняется проверка на корректность значения _value
        if ((_value>=1)&&(_value<=7))
            value = _value;
```



```

    }
    // дополнительный метод
    // возвращает название дня недели, которому соответствует value
    public String GetName() {
        String day = "Monday";
        switch (value) {
            case 2: day = "Tuesday"; break;
            case 3: day = "Wednesday"; break;
            case 4: day = "Thursday"; break;
            case 5: day = "Friday"; break;
            case 6: day = "Saturday"; break;
            case 7: day = "Sunday"; break;
        }
        return day;
    }
}

```

Доступ к элементам класса, которые объявлены с модификатором доступа `protected`.

Рассматривают два случая доступа к `protected`-элементу класса:

- в пределах пакета, в котором класс с данным `protected`-элементом объявлен (в текущем пакете);
- из другого пакета.

Если в некотором пакете элемент класса объявлен с ключевым словом `protected`, то:

- в текущем пакете этот элемент есть видимый из любого кода (также как и `public`-элемент);
- в другом пакете этот элемент есть видимым только в производных классах.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Структура класса в Java.
2. Как создаются методы класса?
3. Каким образом создается экземпляр класса?
4. Синтаксис класса.

ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

Задание 1.

Напишите класс, конструктор которого принимает два массива: массив значений и массив весов значений. Класс должен содержать метод, который будет возвращать элемент из первого массива случайным образом, с учётом его веса. Пример:

Дан массив [1, 2, 3], и массив весов [1, 2, 10].

В среднем, значение «1» должно возвращаться в 2 раза реже, чем значение «2» и в десять раз реже, чем значение «3».

Задание 2.

Вариант 1. Создать класс Dog, в котором описать характеристики и поведение собаки. Вызвать его.

Вариант 2. Создать класс Hotel в котором описать основные характеристики отеля. Вызвать его.

Вариант 3. Создать класс Car в котором описать основные характеристики и действия машины. Вызвать его.

Вариант 4. Создать класс Tree в котором описать основные характеристики дерева. Вызвать его.

Вариант 5. Создать класс Person в котором описать основные характеристики и действия человека. Вызвать его.

Вариант 6. Создать класс Student в котором описать основные характеристики и действия студента. Вызвать его.

Вариант 7. Создать класс Hotel в котором описать основные характеристики отеля. Вызвать его.

Вариант 8. Создать класс Tree в котором описать основные характеристики дерева. Вызвать его.

Вариант 9. Создать класс Hotel в котором описать основные характеристики отеля. Вызвать его.

Вариант 10. Создать класс Person в котором описать основные характеристики и действия человека. Вызвать его.

Вариант 11. Создать класс Car в котором описать основные характеристики и действия машины. Вызвать его.

Вариант 12. Создать класс Dog, в котором описать характеристики и поведение собаки. Вызвать его.

Вариант 13. Создать класс Student в котором описать основные характеристики и действия студента. Вызвать его.

Вариант 14. Создать класс Tree в котором описать основные характеристики дерева. Вызвать его.

Вариант 15. Создать класс Hotel в котором описать основные характеристики отеля. Вызвать его.

Вариант 16. Создать класс Car в котором описать основные характеристики и действия машины. Вызвать его.

Вариант 17. Создать класс Person в котором описать основные характеристики и действия человека. Вызвать его.

Вариант 18. Создать класс Tree в котором описать основные характеристики дерева. Вызвать его.

Вариант 19. Создать класс Dog, в котором описать характеристики и поведение собаки. Вызвать его.

Вариант 20. Создать класс Hotel в котором описать основные характеристики отеля. Вызвать его.

Вариант 21. Создать класс Student в котором описать основные характеристики и действия студента. Вызвать его.

Вариант 22. Создать класс Person в котором описать основные характеристики и действия человека. Вызвать его.

Вариант 23. Создать класс Car в котором описать основные характеристики и действия машины. Вызвать его.

Вариант 24. Создать класс Tree в котором описать основные характеристики дерева. Вызвать его.

Вариант 25. Создать класс Dog, в котором описать характеристики и поведение собаки. Вызвать его.

Вариант 26. Создать класс Hotel в котором описать основные характеристики отеля. Вызвать его.

Вариант 27. Создать класс Student в котором описать основные характеристики и действия студента. Вызвать его

ДОМАШНЕЕ ЗАДАНИЕ

Индивидуальное задание

ЛИТЕРАТУРА

И. Н. Блинов В. С. Романчик, Java, Четыре четверти, 2020.

Преподаватель

А.С.Кибисова

Рассмотрено на заседании цикловой комиссии
программного обеспечения информационных
технологий

Протокол № _____ от «___» _____ 2021

Председатель ЦК _____ В.Ю.Михалевич