

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 № 3  
Инструкционно-технологическая карта

Тема: «Наследование и инкапсуляция на языке Java»

Цель: Научиться работать с наследованием и инкапсуляцией, в том числе с геттерами и сеттерами, научиться их различать, а также научиться работать с модификаторами доступа в языке Java.

Время выполнения: 6 часов

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 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:

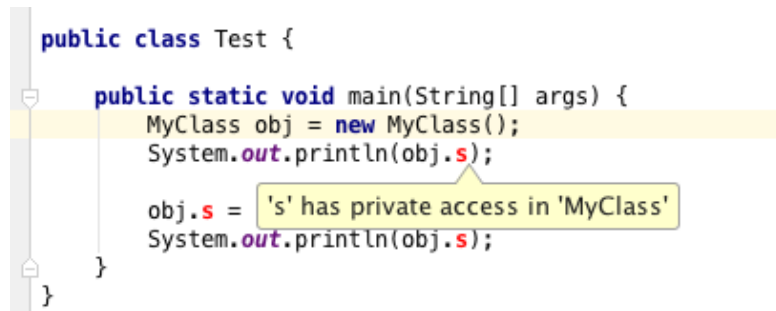
```
Hello World!
```

```
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 объявлены 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. Что такое инкапсуляция?
- 2. Что такое иерархия классов?
- 3. Как наследуются поля и методы классов?
- 4. Охарактеризуйте модификаторы доступа (Access modifiers)
- 5. Охарактеризуйте Геттеры и Сеттеры (Getters and Setters)
- 6. Охарактеризуйте public
- 7. Охарактеризуйте default
- 8. Охарактеризуйте protected
- 9. Охарактеризуйте private
- 

### • ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

Задание 1.

1. Создайте проект Java. Назовите пакет com.example , а главный класс EmployeeTest

2. Создайте пакет com.example.domain, а в нем класс Employee с указанными полями:

```
public int empId;
public String name;
public String ssn;
public double salary;
```

3. Добавьте конструктор класса:

```
public Employee() { }
```

4. Создайте методы чтения и записи ( «геттеры» и «сеттеры») для каждого поля. Используйте для этого контекстное меню редактора Eclipse

5. Добавьте в файл класса EmployeeTest импорт класса Employee

```
import com.example.domain.Employee;
```

6. Добавьте в процедуру main класса EmployeeTest команды создания объекта класса

```
Employee и заполнение его полей
Employee emp = new Employee();
emp.setEmpId(101);
emp.setName("Jane Smith");
emp.setSalary(120_345.27);
emp.setSsn("012-34-5678");
```

7. Добавьте в процедуру main класса EmployeeTest команды отображения данных объекта класса Employee

```
System.out.println("Employee ID: "+emp.getEmpId());
System.out.println("Employee Name: "+emp.getName());
System.out.println("Employee Soc Sec # "+emp.getSsn());
System.out.println("Employee salary: "+emp.getSalary());
```

8. Запустите приложение.

## Задание 2.

- Вариант 1. Сотрудник, 3 поля – 3 класса наследника
- Вариант 2. Студент, 2 поля – 2 класса наследника
- Вариант 3. Товар, 3 поля– 4 класса наследника
- Вариант 4. Собака, 2 поля– 3 класса наследника
- Вариант 5. Геометрическая фигура, 3 поля– 3 класса наследника
- Вариант 6. Программное обеспечение, 4 поля– 2 класса наследника
- Вариант 7. Аппаратное обеспечение, 3 поля– 3 класса наследника
- Вариант 8. Город, 2 поля– 2 класса наследника
- Вариант 9. Страна, 2 поля– 2 класса наследника
- Вариант 10. Книга, 3 поля – 4 класса наследника
- Вариант 11. Кот, 2 поля– 3 класса наследника
- Вариант 12. Товар, 3 поля– 4 класса наследника
- Вариант 13. Сотрудник, 3 поля – 3 класса наследника
- Вариант 14. Страна, 2 поля– 3 класса наследника
- Вариант 15. Геометрическая фигура, 3 поля– 3 класса наследника
- Вариант 16. Программное обеспечение, 4 поля– 3 класса наследника
- Вариант 17. Сотрудник, 3 поля – 3 класса наследника
- Вариант 18. Город, 2 поля– 3 класса наследника
- Вариант 20. Товар, 3 поля– 4 класса наследника
- Вариант 21. Аппаратное обеспечение, 3 поля– 4 класса наследника
- Вариант 22. Студент, 2 поля – 2 класса наследника
- Вариант 23. Собака, 2 поля– 3 класса наследника
- Вариант 24. Книга, 3 поля – 3 класса наследника
- Вариант 25. Сотрудник, 3 поля – 3 класса наследника
- Вариант 26. Кот, 2 поля– 2 класса наследника
- Вариант 27. Товар, 3 поля– 4 класса наследника

## 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