

# Java Professional module #1

lecture#1 Classes, Object, Modifier, Encapsulation  
Mentor: Iurii Avramenko

## lecture#1 Classes, Object, Modifier, Encapsulation

### Классы и объекты

- Что такое объект и класс в Java?
- Создание класса
- Создание объекта
- Конструктор
- Доступ к переменным экземпляра и методам
- Правила объявления классов, операторов импорта и пакетов в исходном файле

### Модификаторы доступа

#### Геттеры и сеттеры

- Что такое геттеры и сеттеры?
- Зачем нужны геттеры и сеттеры?
- Правила именования геттеров и сеттеров

### Инкапсуляция

- Что такое инкапсуляция
- Преимущества инкапсуляции

## Объекты и классы в Java

Java является объектно-ориентированным языком программирования.

Как язык, который имеет функцию объектно-ориентирования, он поддерживает следующие основные понятия:

- полиморфизм;
- наследование;
- инкапсуляция;
- абстракция;
- классы;
- объекты;
- экземпляр;
- метод.

**Класс** может быть определен как **шаблон**, который описывает **поведение** объекта, который в свою очередь имеет состояние и поведение. Объект является экземпляром класса.

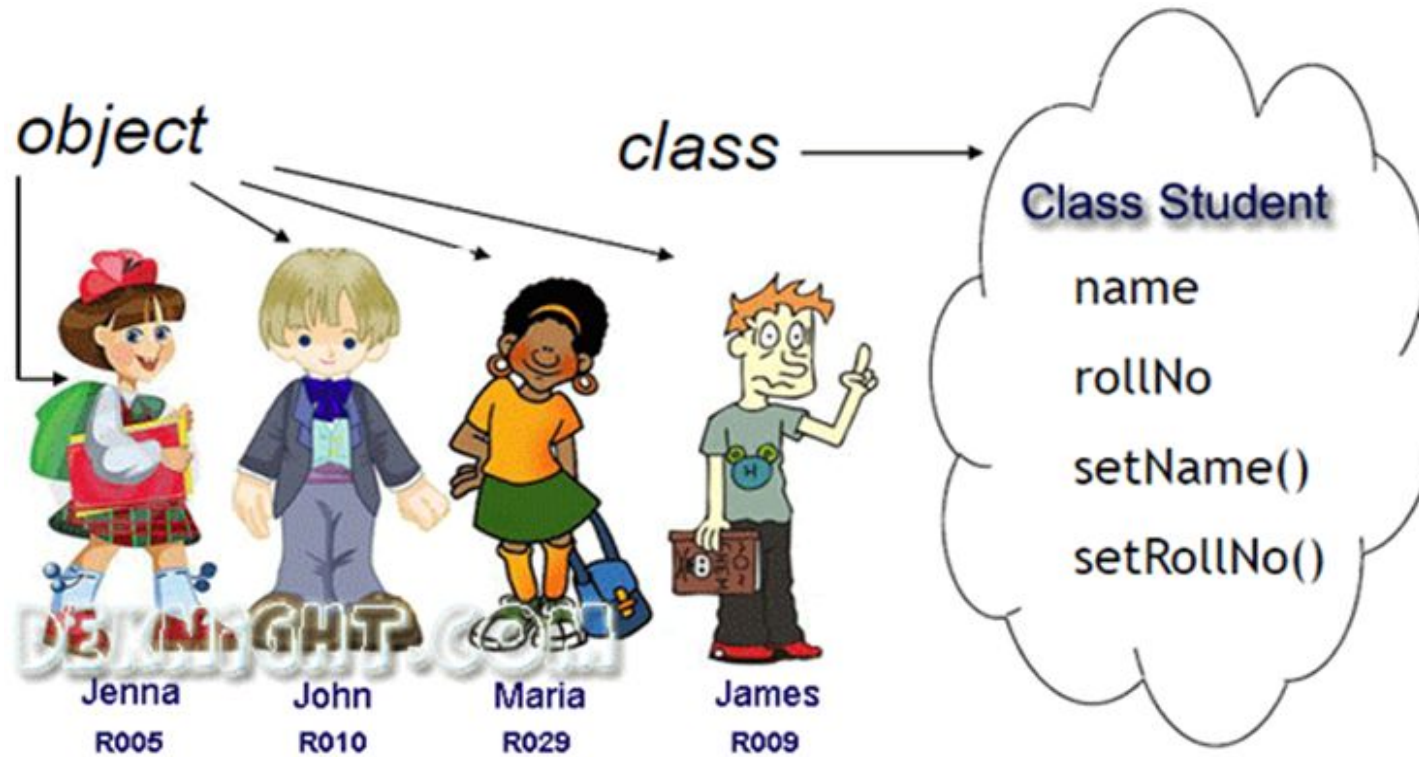
Например:

собака может иметь состояние – цвет, имя, а также и поведение – кивать, лаять, есть.

## Что такое объект и класс в Java?

- Класс в Java - это шаблон для создания объекта.
- Объект - это экземпляр класса.
- Класс определяет структуру и поведение, которые будут совместно использоваться объектами.
- Класс содержит переменные и методы, которые называются элементами класса, членами класса.
- Класс составляет основу инкапсуляции в Java.
- Каждый объект данного класса содержит структуру и поведение, которые определены классом.

Пример:



Объявлен класс `Student`, у которого есть:

- переменные `name` и `rollNo`,
- методы `setName()` и `setRollNo()` для установки этих значений.

На основе этого класса создано несколько объектов: Jenna, John, Maria, James.

У каждого объекта, то есть студента, есть `name` и `rollNo`, но они разные!

## Создание класса в Java

### Синтаксис

```
public class Box {  
    double width;  
    double height;  
    double depth;  
}
```

После ключевого слова `class` пишется имя класса.

**Имя класса с Заглавной буквы – всегда!!!**

В теле класса объявляются переменные и методы класса.  
Их может быть сколько угодно.

```
<modifier> class <Name> {
```

```
    тип переменная1;  
    тип переменная2;  
    тип переменнаяN;
```

```
    тип имяМетода1 ( список параметров ) {  
        // тело метода  
    }
```

```
    тип имяМетода2 (список параметров) {  
        // тело метода  
    }
```

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

## Создание объекта в Java

При создании экземпляра класса, создается объект, который содержит собственную копию каждой переменной экземпляра, определенной в данном классе.

Создание объектов класса представляет собой двух-этапный процесс:

### 1. *Объявление переменной типа класса.*

Эта переменная не определяет объект.

Это указатель, который может ссылаться на объект.

Например: `String str;`

**Объявление класса создает только шаблон, но не конкретный объект.**

### 2. *Создание объекта.*

С помощью оператора *new* динамически (то есть во время выполнения) резервируется память для объекта и возвращается ссылка на него: `str = new String();`

**Чтобы создать объект** класса `Box` в Java, нужно воспользоваться оператором `new`: `-> Box myBox = new Box();`

## Конструктор класса


- **Каждый** класс имеет конструктор.
- Если мы не напишем его или, например, забудем, компилятор создаст его по умолчанию для этого класса.
- Каждый раз, когда в Java создается новый объект, будет вызываться по меньшей мере один конструктор.
- Главное правило является то, что они должны иметь то же имя, что и класс.

Синтаксис:

```
public class Puppy {  
    String name;  
  
    public Puppy() {} // конструктор «по умолчанию» без параметров  
  
    public Puppy(String name) { // конструктор с параметрами  
        this.name = name;  
    }  
}
```

Вопрос ??? Сколько конструкторов может быть у класса?





Доступ к переменным экземпляра и методам


Переменные и методы доступны в Java через созданные объекты.

Чтобы получить доступ к переменной экземпляра, полный путь должен выглядеть следующим образом:

```
/* Сначала создайте объект */  
ObjectReference = new Constructor();
```

```
/* Теперь вызовите переменную следующим образом */  
ObjectReference.variableName;
```

```
/* Теперь Вы можете вызвать метод класса */  
ObjectReference.methodName();
```



## Правила объявления классов, операторов импорта и пакетов в исходном файле

- В исходном файле может быть только один публичный класс (public class).
- Исходный файл может иметь несколько "непубличных" классов.
- Название публичного класса должно совпадать с именем исходного файла, который должен иметь расширение **.java** в конце.  
Например: имя класса *public class Employee{}*, то исходный файл должен быть *Employee.java*.
- Если класс определен внутри пакета, то оператор пакет должно быть первым оператором в исходном файле.
- Если присутствуют операторы импорта, то они должны быть написаны между операторами пакета и объявлением класса. Если нет никаких операторов пакета, то оператор импорта должен быть первой строкой в исходном файле.
- Операторы импорта и пакета будут одинаково выполняться для всех классов, присутствующих в исходном файле. В Java не представляется возможным объявить различные операторы импорта и/или пакета к различным классам в исходном файле.

## Модификаторы доступа

Все члены класса в языке Java - поля и методы - имеют модификаторы доступа.

Модификаторы доступа позволяют задать допустимую область видимости для членов класса, то есть контекст, в котором можно употреблять данную переменную или метод.

Modifier	Class	Package	Subclass	Global
Public	Yes	Yes	Yes	Yes
Protected	Yes	Yes	Yes	No
Default	Yes	Yes	No	No
Private	Yes	No	No	No

## Виды модификаторов доступа

В Java используются следующие модификаторы доступа:

**public:** публичный, общедоступный класс или член класса.

Поля и методы, объявленные с модификатором `public`, видны другим классам из текущего пакета и из внешних пакетов.

**private:** закрытый класс или член класса, противоположность модификатору `public`.

Закрытый класс или член класса доступен только из кода в том же классе.

**protected:** такой класс или член класса доступен из любого места в текущем классе или пакете или в производных классах, даже если они находятся в других пакетах

**default:** Отсутствие модификатора у поля или метода класса предполагает применение к нему модификатора по умолчанию. Такие поля или методы видны всем классам в текущем пакете.




Что такое геттеры и сеттеры? `get()`, `set()`

В Java геттер и сеттер — это два обычных метода, которые используются для получения значения поля класса или его изменения.

`set()` — это метод, который изменяет (устанавливает; от слова *set*) значение поля.

`get()` — это метод, который возвращает (от слова *get*) нам значение какого-то поля.

Геттер называют *accessor* (аксессор, т.к. он предоставляет доступ к полю).  
Сеттер *mutator* (мутатор, т.к. он меняет значение переменной).



## Зачем нужны геттеры и сеттеры?

1. помогают достичь инкапсуляции для скрытия состояния объекта и предотвращения прямого доступа к его полям
2. при реализации только геттера (без сеттера) можно достичь неизменяемости объекта
3. они могут предоставлять дополнительные функции:
  - проверка корректности значения перед его присваиванием полю или обработка ошибок.
  - мы можем добавить условную логику и обеспечить поведение в соответствии с потребностями.

*если сеттер не имеет подобной логики, а лишь присваивает полю какое-то значение, то его наличие не обеспечивает инкапсуляцию. А его присутствие становится фиктивным.*
4. можем предоставить полям разные уровни доступа:  
например, *get* (доступ для чтения) может быть *public*,  
*set* (доступ для записи) может быть *protected*
5. с их помощью мы достигаем еще одного ключевого принципа ООП — абстракции, которая скрывает детали реализации, чтобы никто не мог использовать поля непосредственно в других классах или модулях

## Правила именования геттеров и сеттеров

Поле	Геттер	Сеттер
int quantity	int getQuantity()	void setQuantity(int quantity)
String name	String getName()	void setName(String name)
Date birthday	Date getBirthday()	void setBirthday(Date birthday)
boolean rich	boolean isRich()	void setRich(boolean rich)



Что такое инкапсуляция


В программировании есть два распространенных понятия — **инкапсуляция** и **сокрытие**.

Значение слова «*инкапсуляция*» в программировании — **объединение данных и методов работы с этими данными в одной упаковке** («капсуле»).

В Java в роли упаковки-капсулы выступает *класс*.

Класс содержит в себе и *данные* (поля класса), и *методы* для работы с этими данными.

С сокрытием данных нам помогают:

1. модификаторы доступа (*private, protected, package default*);
  2. геттеры и сеттеры: *get(), set()*;
- 



## Важные преимущества инкапсуляции

### 1. **Контроль за корректным состоянием объекта.**

Благодаря сеттерам и модификаторам `private`, мы можем обезопасить нашу программу.

### 1. **Удобство для пользователя за счет интерфейса.**

Мы оставляем «снаружи» для доступа пользователя только методы.

Ему достаточно вызвать их, чтобы получить результат, и совсем не нужно вникать в детали их работы.

### 1. **Изменения в коде не отражаются на пользователях.**

Все изменения мы проводим внутри методов. На пользователя это не повлияет:

Например: мы создаем метод `gas()`, для класса `Car`.

Пользователь как писал `auto.gas()` для газа машины, так и будет писать.

А то, что мы поменяли что-то в работе метода `gas()` для него останется незаметным: он, как и раньше, просто будет получать нужный результат.