



# Принципы ООП

Источник: <https://training.by/#!/News/275?lang=ru>



# Инкапсуляция

Одним из самых важных факторов при проектировании компонентов приложения является сокрытие внутренних данных компонента и деталей его реализации от других компонентов приложения и предоставление набора методов для взаимодействия с ним (API).



## Правильная инкапсуляция важна по многим причинам:

1. Она способствует переиспользованию компонентов: поскольку в этом случае компоненты взаимодействуют друг с другом только посредством их API и безразличны к изменениям внутренней структуры, они могут использоваться в более широком контексте.
2. Инкапсуляция ускоряет процесс разработки: слабо связанные друг с другом компоненты (то есть компоненты, чей код как можно меньше обращается или использует код других компонентов) могут разрабатываться, тестироваться и дополняться независимо.
3. Правильно инкапсулированные компоненты более легки для понимания и процесса отладки, что упрощает поддержку приложения.

В языке Java инкапсуляция реализована с помощью **системы классов**, которые позволяют собрать информацию об объекте в одном месте; **пакетов**, которые группируют классы по какому-либо критерию, и **модификаторов доступа**, которыми можно пометить весь класс или его поле или метод.



## Всего модификаторов доступа четыре:

**public** – полный доступ к сущности (полю или методу класса) из любого пакета;

**protected** – доступ к сущности только для классов своего пакета и наследников класса;


**неявный модификатор по умолчанию** (при отсутствии трёх явных) – доступ к сущности только для классов своего пакета;

**private** – доступ только внутри класса, в котором объявлена сущность.



## Наследование

Наследование является одним из важнейших принципов объектно-ориентированного программирования, поскольку оно позволяет создавать иерархические структуры объектов. Используя наследование, можно создать общий класс, который будет определять характеристики и поведение, свойственные какому-то набору связанных объектов. В дальнейшем этот класс может быть унаследован другими, более частными классами, каждый из которых будет добавлять уникальные, свойственные только ему характеристики и дополнять или изменять поведение базового класса. В терминах Java такой общий класс называется **суперклассом** (superclass), или **базовым классом** (base class), или **классом-родителем** (parent class), а класс, его наследующий, - **подклассом** (subclass), или **дочерним классом** (child class), или **классом-потомком** (derived class).




```
class Employee {  
    private String firstName;  
    private String lastName;  
    private double salary;  
    private LocalDate hireDate;  
  
    //constructors  
  
    public double getSalary() {  
        return salary;  
    }  
}  
  
class Manager extends Employee {  
    private double bonusPercentage;  
  
    //constructors  
  
    public double getSalary() {  
        return super.salary * (100 + bonusPercentage) / 100;  
    }  
}
```



## Полиморфизм

Полиморфизм - один из принципов ООП, позволяющий вызовом переопределённого метода через переменную класса-родителя получить поведение, которое будет соответствовать реальному классу-потомку, на который ссылается эта переменная.



```
abstract class Vehicle {
    abstract void start();
}

class SportCar extends Vehicle {
    void start() {
        System.out.println("Starting my fancy sport car!");
    }
}

class Truck extends Vehicle {
    void start() {
        System.out.println("Starting my heavy truck!");
    }
}
```

```
Vehicle vehicle = new SportCar();
vehicle.start();
vehicle = new Truck();
vehicle.start();
```





## Абстракция

**Абстракция** (от лат. abstractio — выделение, отвлечение или отделение) — теоретический прием исследования, позволяющий отвлечься от некоторых несущественных в определенном отношении свойств изучаемых явлений и выделить свойства существенные и определяющие.