@Override и переопределение методов

Ярким представителем механизма наследования является аннотация `@Override` .

Мы уже сталкивались с этим флагом. Сегодня разберемся, что же конкретно он маркирует.

> В Java вам абсолютно необязательно пользоваться именно той версией метода, что описана в родителе. Если у вас есть доступ к этому методу (он объявлен в родителе не как `private`), вы можете его **переписать**.

Чтобы переписать метод, вам нужно объявить точно такой же (по сигнатуре) метод в вашем наследнике и пометить его флагом `@Override` . Это позволит IDEA и любым читателям вашего кода (вам в том числе) сразу понять, что этот метод есть где-то в родителях, а вы его просто переписываете.

> Переписывание метода в Java называется **переопределением**.

Рассмотрим пример:

```
public class A {
   public void doSmth() {
      System.out.println("A");
   }
}
```

```
public class B extends A {

   @Override
   public void doSmth() {
      System.out.println("B");
   }

   public void doSmth2() {
   // Так как doSmth2 нет в A, мы не должны помечать его @Override
      System.out.println("B2");
   }
}

public class Main1 {
   public static void main(String[] args) {
      A a = new A();
      A b1 = new B();
      B b2 = new B();

      a.doSmth(); // Выведет A
      b1.doSmth(); // Выведет B
      b1.doSmth2(); // Будет ошибка
                    // Так как метода doSmth2() нет в классе A
                    // и ссылка типа A о нем не знает
      b2.doSmth(); // Выведет B
      b2.doSmth2(); // Выведет B2
   }
}
```

Переопределять методы обязательно:

когда вам нужно изменить их логику (родительская логика не подходит);

когда переписываемый метод в объекте является абстрактным (то есть не имеет тела).

Перегрузка методов

В Java есть схожее с переопределением метода понятие — **перегрузка метода**.

Перегрузка метода — это прием, который позволяет вызывать один и тот же метод с разными параметрами и получить совершенно разную обработку этих параметров.

Этот прием используется редко, но его надо четко отличать от **переопределения метода**.

Представим, что нам нужно написать метод, который применяет разную логику в зависимости от того, какой поступил тип данных. Например, нам нужно получить число и напечатать его уже в квадрате (во второй степени). Есть два варианта того, как можно это реализовать:

Объявить один метод с двумя параметрами и всегда передавать в него один из них.

Перегрузить метод, написав две его вариации под разные типы параметров.

Первый вариант

В случае, когда нам нужно передать строку, нам придется также передать какое-то дефолтное значение в метод для числа — чтобы показать методу, что он **не должен работать с этим числом**. Таким значением может быть `-1`. В случае с отсутствием строки дефолтным значением будет `null`.

```
public class Main2 {
   public static void main(String[] args) {
      int n1 = 5;
      String n2 = "6";
      printSquare(n1, null); // Напечатает 25
      printSquare(-1, n2); // Напечатает 36
   }

   public static void printSquare(int num, String numStr) {
      if (num != -1) {
         System.out.println(num * num);
      } else {
         int parsedNum = Integer.parseInt(numStr);
         System.out.println(parsedNum * parsedNum);
      }
   }
}
```

Если пользователь этого приложения забудет, что нужно передавать **−1** — всё сломается.

Второй вариант

Мы можем объявить два одинаковых метода (одно имя, один возвращаемый тип), но с разным набором параметров.

Это и называется **перегрузкой**, или **overload**.

```
public class Main3 {
    public static void main(String[] args) {
        int n1 = 5;
        String n2 = "6";

        printSquare(n1); // Напечатает 25
        printSquare(n2); // Напечатает 36
    }

    public static void printSquare(int num) {
        System.out.println(num * num);
    }

    public static void printSquare(String str) {
        int parsedNum = Integer.parseInt(str);
        System.out.println(parsedNum * parsedNum);
    }
}
```