

7.2

Примитивы и объекты

00:00–01:31

Привет! В этом видео мы изучим так называемые примитивы и сравним их с объектами.

Примитивы бывают восьми видов.

Примитивы

- `boolean`
- `byte`
- `char`
- `short`
- `int`
- `long`
- `float`
- `double`

Со всеми видами примитивов мы познакомимся в этом модуле.

Итак, что же такое примитив? Примитив — это, по сути, самый простой тип данных в Java. Примитив — от слова «примитивный», то есть простой.

Давайте на примере посмотрим, что это такое.

Мы уже знаем такой тип данных, как объект. Вот, например, у нас есть класс `product`, объекты которого — это некие продукты, содержащие в себе названия и цены. Цена — целое число `int`, с которым вы хорошо знакомы — это и есть примитив `A`. Название — это объект класса `string`, которое примитивом не является.

```
private String name;  
private int price;
```

01:31–04:06

Давайте сравним примитивы с объектами и посмотрим, чем они отличаются и чем похожи.

Первое отличие заключается в том, что у объекта есть некое состояние, значение всех его переменных и поведение — методы класса, которые позволяют работать с этим объектами, а у примитивов есть только значение.

Второе отличие состоит в том, что переменная-объект — это ссылка на этот объект в памяти, а примитив содержит просто значение, то есть это не ссылка на него.

Мы уже изучали, что переменные-объекты — это ссылки. Давайте вспомним, как они работают и чем отличаются от примитивов.

Пусть у нас есть некий продукт `milk`. Это будет переменная, которая содержит в себе ссылку на этот объект памяти.

```
Product milk = new Product (name: "молоко", price: 75);
```

Если мы напишем элемент `milk 2`, он тоже будет ссылаться на тот же самый объект памяти.

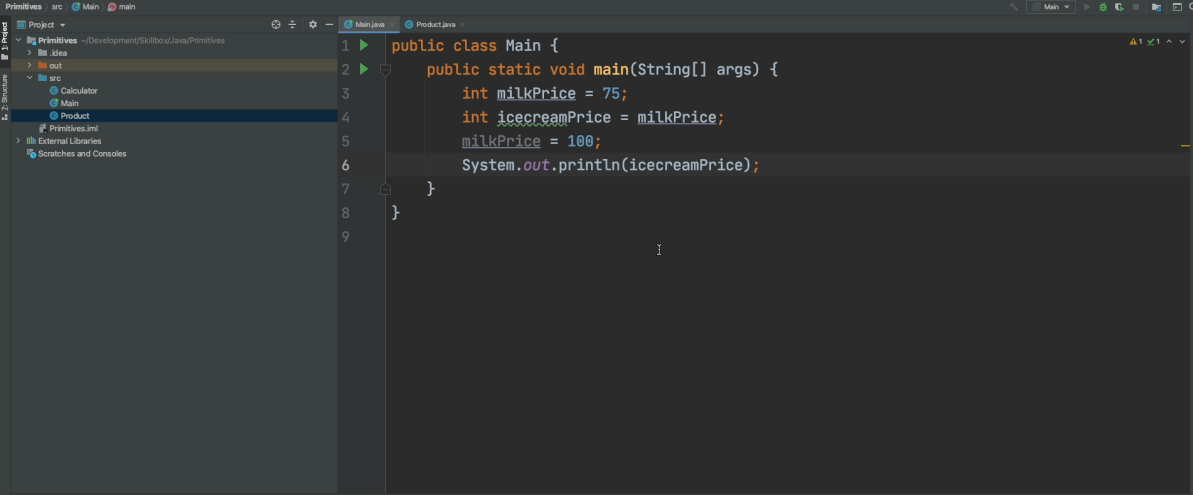
```
Product milk2 = milk;
```

Если мы возьмём и выведем этот объект в консоль, то увидим, что это он же.

Как мы можем быть уверены, что здесь объект не скопировался, а скопировалась только ссылка на него? Мы можем взять и поменять значение цены в объекте milk. Например, написать в следующей `setPrice (100)` и по-прежнему вывести здесь объект milk 2 .

Мы можем увидеть, что в milk 2 эта цена тоже поменялась, поскольку и milk и milk 2 — это переменная, содержащая в себе ссылку на один и тот же объект в памяти.

Теперь посмотрим на примитив. Зададим некое значение `int`, пусть это будет цена молока, которая равняется тем же 75 рублям. Создадим новую переменную, например цену мороженого, и приравняем её цену к цене молока. Теперь, если мы поменяем цену молока в следующей строке и выведем значение цены мороженого в консоль, то увидим, что цена мороженого не изменится: она по-прежнему останется равной 75 рублям.



```
1 public class Main {
2     public static void main(String[] args) {
3         int milkPrice = 75;
4         int icecreamPrice = milkPrice;
5         milkPrice = 100;
6         System.out.println(icecreamPrice);
7     }
8 }
9
```

The screenshot shows the IntelliJ IDEA IDE with a project named 'Primitives'. The 'src' directory contains 'Main.java'. The code in 'Main.java' is as follows:

```
1 public class Main {
2     public static void main(String[] args) {
3         int milkPrice = 75;
4         int icecreamPrice = milkPrice;
5         milkPrice = 100;
6         System.out.println(icecreamPrice);
7     }
8 }
9
```

The 'Run' console at the bottom shows the command: `/Library/Java/JavaVirtualMachines/jdk-11.0.1.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/lib`. The status bar at the bottom indicates 'Build completed successfully in 1 s 349 ms (moments ago)'.

Это происходит из-за того, что в переменной-примитиве хранится не ссылка, а значение. Если приравнять новую переменную-примитив к

уже существующей, то это значение просто копируется. Несмотря на то что переменные-примитивы содержат в себе просто значения, а переменные-объекты — ссылки, работают они похожим образом, поскольку на самом деле и те и другие содержат значения: первые содержат просто значение, а вторые содержат некие специальные числа, которые указывают на расположение объектов памяти.

Чтобы убедиться в том, что они работают похожим образом, давайте посмотрим, как долго хранятся в памяти ссылки на объекты и как долго хранятся в памяти примитивы.

04:05–05:55

Например, у нас есть некий класс «калькулятор», который производит вычисления с переданными в него числами — например, умножает эти числа друг на друга. Напишем метод, в котором будет создаваться объект этого класса, например метод Calculate.

```
public static int calculate(int a, int b, int c)
Calculator calculator = new Calculator(a, b, c);
return calculator.multiply();
```

Если потом ссылка на этот объект — объект класса «калькулятор» — нигде больше не будет использоваться и если мы вызовем этот метод...

```
System.out.println(calculate(a:5, b:10, c:15));
```

...то ссылка на объект класса «калькулятор» будет только внутри этого метода, и по завершении этого метода ссылок на этот объект не останется. Через некоторое время этот объект будет автоматически удалён из памяти.

Если же мы вернём этот объект из метода — для этого метод нужно переименовать, чтобы его имя соответствовало назначению...

```
public static int Calculator getCalculator (int a, int b,
int c)
Calculator calculator = new Calculator(a, b, c);
return calculator;
```

...заменим вызванный метод...

```
Calculator calculator = getCalculator (a:5, b:10, c:15);
System.out.println(calculator.multiply());
```

...тогда здесь будет возвращена ссылка на объект с помощью метода, и она скопируется в новую переменную — объект не будет удалён из памяти.

05:55–08:14

Важный момент: пока мы можем обращаться к объекту и пока у него есть хотя бы одна ссылка, объект будет храниться в памяти, но если метод, в котором объект завершился, никуда не передаёт ссылку на этот объект, то ссылок на этот объект больше не останется, и объект будет автоматически удалён из памяти.

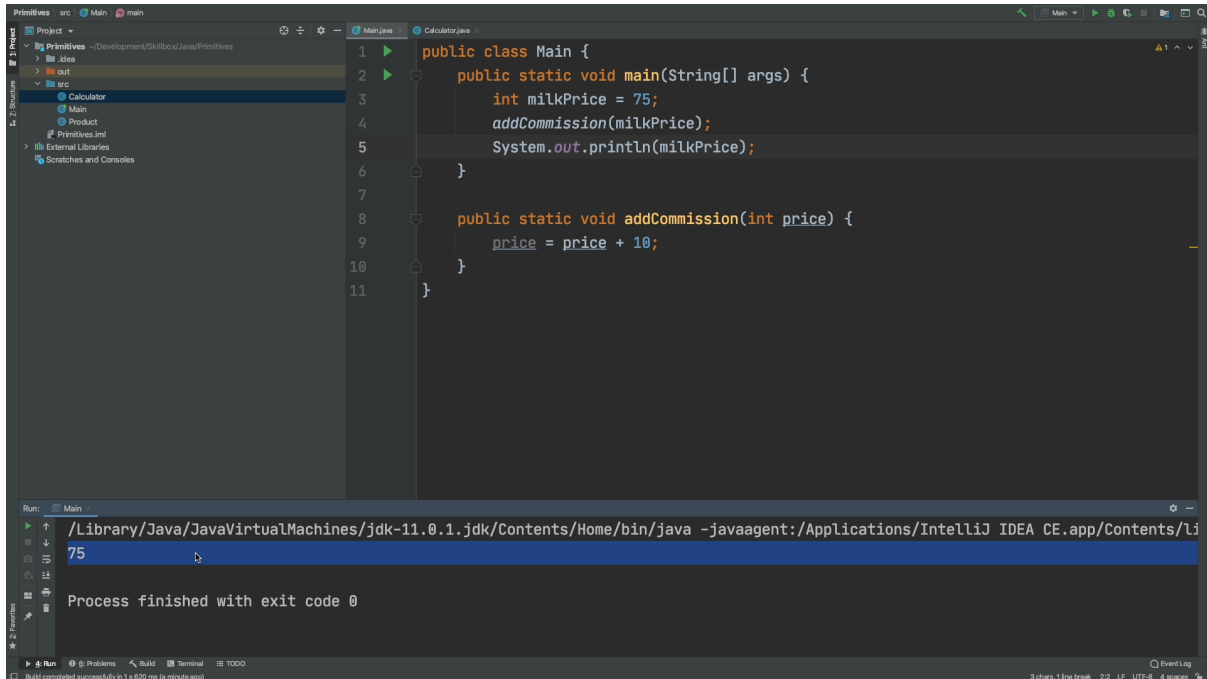
С примитивами происходит аналогичная история: их значение, так же как и ссылка на объект, копируется при передаче в метод или при возвращении из методов.

Давайте посмотрим на примере. Создадим для этого метод, меняющий значение примитива — например, добавляющий некую комиссию к цене.

```
public static void addComission (int price) {
```

```
price = price + 10;  
}
```

Посмотрим, что будет, если использовать этот метод.

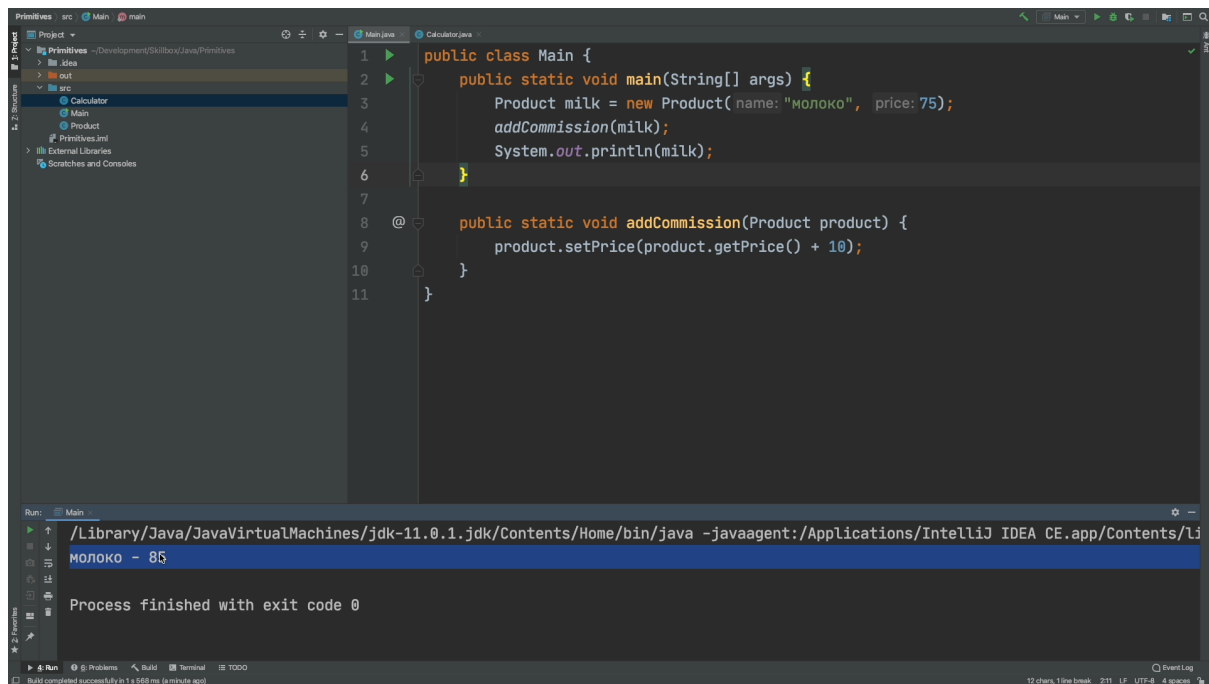


The screenshot shows an IDE with a project named 'Primitives'. The 'Main.java' file is open, showing the following code:

```
1 public class Main {  
2     public static void main(String[] args) {  
3         int milkPrice = 75;  
4         addCommission(milkPrice);  
5         System.out.println(milkPrice);  
6     }  
7  
8     public static void addCommission(int price) {  
9         price = price + 10;  
10    }  
11 }
```

The 'Run' window at the bottom shows the command executed: `/Library/Java/JavaVirtualMachines/jdk-11.0.1.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/Li`. The output is `75`, and the message 'Process finished with exit code 0' is displayed.

Получилось снова 75. Мы видим, что значение примитива не изменилось. Это произошло именно потому, что значение скопировалось при передаче в метод. Если бы мы передали в метод переменную-объект, например объект класса `product` с такой же ценой, и метод принимал бы в качестве параметра объект и увеличивал бы его цену, то поменялось бы значение внутри объекта.



При этом значение переменной, то есть ссылка на этот объект, на самом деле тоже скопировалось.

Таким образом, несмотря на то что переменные-объекты — это ссылки, а переменные-примитивы — это значения, работают они похожим образом.

08:14–10:19

Следующее отличие объектов и примитивов — это механизмы сравнения. Сравнивать примитивы с помощью двойного неравенства можно, поскольку они содержат в себе значения. Если мы напишем...

```
int milkPrice = 75;
int icecreamPrice = 75;
if (milkPrice == icecreamPrice) {
    System.out.println("Цены одинаковые");
}
```

...то условие внутри оператора if всегда будет выполняться,

поскольку цены молока и мороженого равны.

Если же сравнивать с помощью двойного «равно» объекты, пусть даже одинаковые по содержанию, то сравниваться будут не объекты, а ссылки на них, содержащиеся в этих переменных.

```
Product product1 = new Product(name: "молоко", price 75);
Product product2 = new Product(name: "молоко", price 75);
if(product1 ==product2) {
System.out.println("Одинаковые");
}
```

В данном случае это будут разные объекты в памяти, и ссылки на них тоже будут разные, поэтому условия внутри оператора if выполняться не будут.

Если же скопировать ссылку, то условия внутри оператора if выполнятся, поскольку и product1, и product2 будут содержать в себе одно и тоже значение — значение ссылки на один и тот же объект.

```
Product product1 = new Product(name: "молоко", price 75);
Product product2 = product1;
if(product1 ==product2) {
System.out.println("Одинаковые");
}
```

Таким образом, сравнение переменных-примитивов с помощью двойного «равно» — это сравнение значений этих примитивов, а сравнение объектов — это сравнение их ссылок.

В связи с этим сравнения двух одинаковых по содержанию объектов может привести к тому, что условие if не будет выполняться.

10:19–12:06

И последнее: переменные-объекты могут быть не заданы и могут быть равны специальному значению `null`, а переменные-примитивы не могут быть не заданы: они всегда должны быть заданы, и их значение всегда должно быть отлично от `null`.

Например, если вы напишете...

```
Product milk;  
System.out.println(milk);
```

...то компилятор подчеркнёт такой код и потребует инициализировать эту переменную. Даже если вы обманете компилятор и зададите эту переменную на уровне класса, то значение этой переменной будет равно `null`. Если вы попытаетесь вызвать у такого объекта какой-нибудь метод, то произойдёт ошибка или так называемое исключение `NullPointerException`, что в переводе с английского значит «пустой указатель», или «указатель, равный `null`».

Если же вы не зададите примитив, то ничего страшного не произойдёт: выведется значение этого примитива по умолчанию.

Все примитивы при создании автоматически заполняются значениями по умолчанию, а переменные-объекты нужно обязательно инициализировать, потому что изначально они равны `null`.

Важный параметр: если вы зададите переменную не как параметр класса, а как локальную переменную, то в этом случае компилятор также подчеркнёт такой код как ошибочный. Заполнение примитивов значениями по умолчанию происходит только если этот примитив — параметр класса.

12:06–12:22

Итоги

Итак, в этом видео вы узнали, что такое примитивы, чем они похожи на объекты и чем они от них отличаются. В следующем видео мы кратко разберём все восемь разновидностей примитивов.