

## 7.7

### Классы-обёртки

**00:00–00:44**

#### Введение

Привет! В этом видео мы поговорим о специальных классах-обёртках, которые в Java существуют для всех примитивов и преобразований примитивов в объекты таких классов и обратно.

Итак, в Java для всех примитивов есть специальные классы и любое значение можно представить не только как примитив, но и как объект этого класса.

### Классы-обёртки

Примитив	Класс-обёртка
boolean	Boolean
byte	Byte
char	Character
short	Short
int	Integer
long	Long
float	Float
double	Double

В большинстве случаев имена таких классов совпадают с названиями примитивов, за исключением примитива `char` (для него этот класс называется `character`) и примитива `int`, для которого этот класс называется `integer`. Чуть позже мы поговорим о том, почему их называют классами-обёртками, а пока давайте разберёмся, для чего они нужны и какие у них есть особенности.

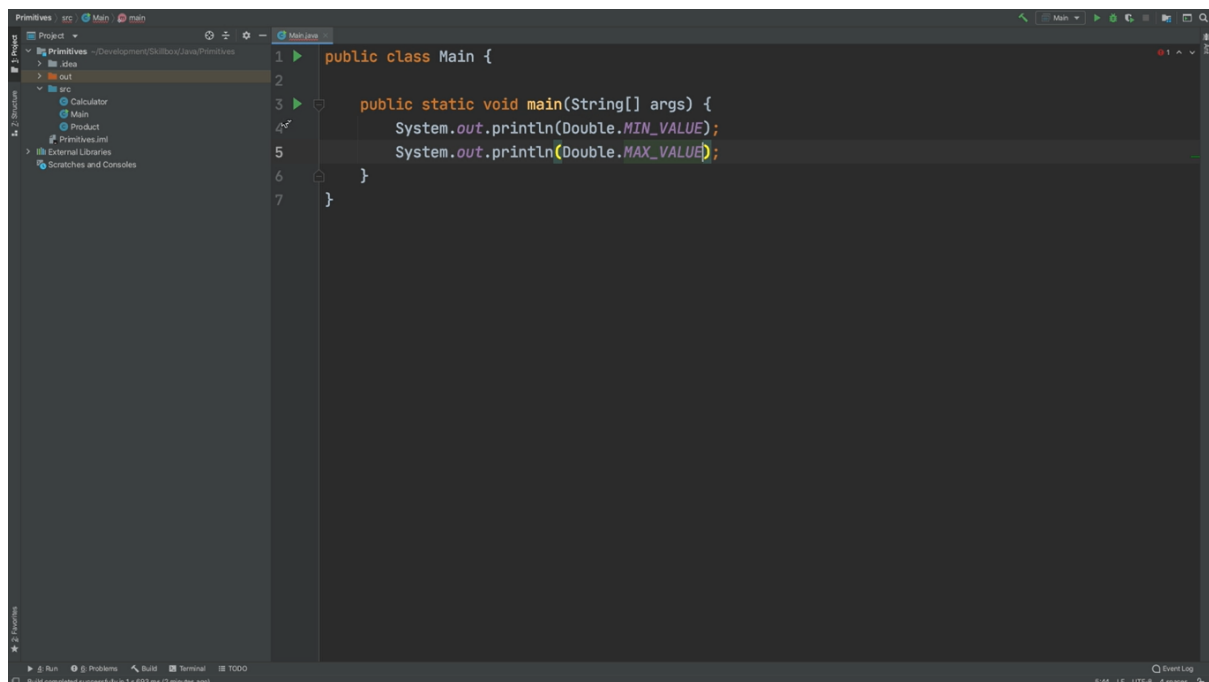
**00:44–02:35**

Особенности классов-обёрток

Во-первых, их экземпляры — это объект. К примеру, переменная класса `integer`, в отличие от переменной типа `int`, может быть равна `null`.

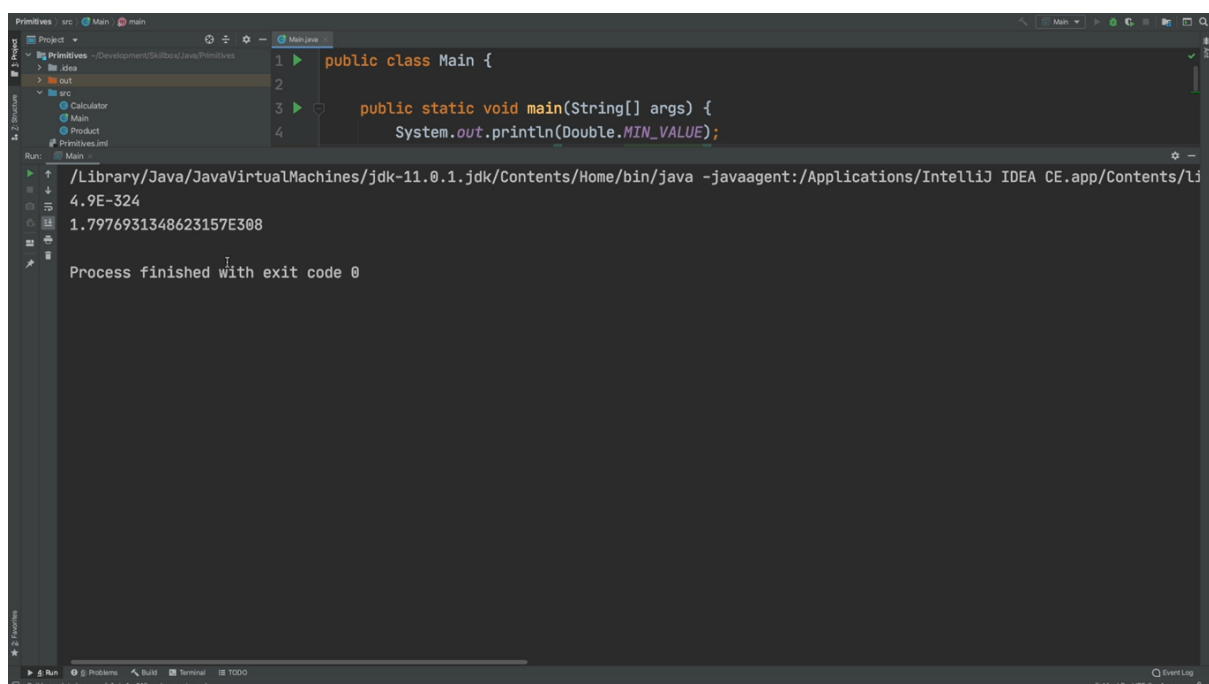
Во-вторых, у классов-обёрток есть большое количество полезных методов и переменных. Например, у каждого такого класса, который является числом, есть переменные, содержащие максимальное и минимальное значения.

Давайте рассмотрим это на примере класса `double`.



```
1 public class Main {
2
3     public static void main(String[] args) {
4         System.out.println(Double.MIN_VALUE);
5         System.out.println(Double.MAX_VALUE);
6     }
7 }
```

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



```
1 public class Main {
2
3     public static void main(String[] args) {
4         System.out.println(Double.MIN_VALUE);
5     }
6 }
```

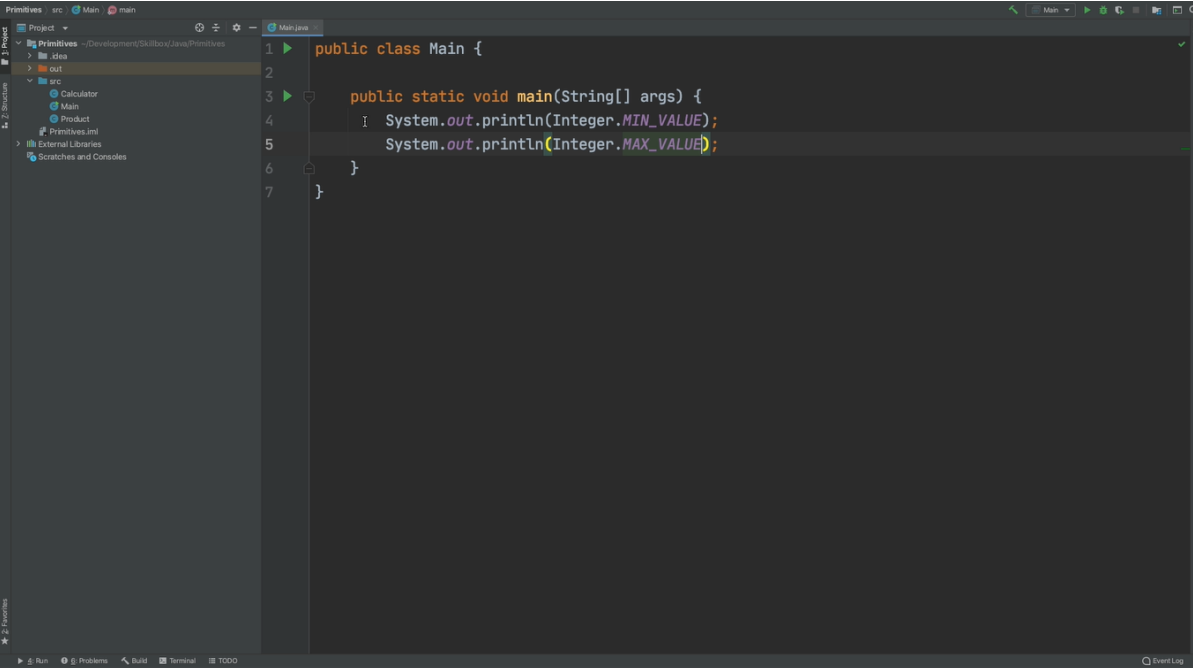
The screenshot shows the IntelliJ IDEA IDE with the same project and code as the previous screenshot. The 'Run' tab is active, showing the output of the program:

```
/Library/Java/JavaVirtualMachines/jdk-11.0.1.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/Li
4.9E-324
1.7976931348623157E308
Process finished with exit code 0
```

При запуске вы можете увидеть максимальное и минимальное значения —  $4,9 * 10^{324}$  и  $1,7 * 10^{308}$ .

Обратите внимание, что для чисел с плавающей точкой это положительное значение. Мы знаем, что у чисел с плавающей точкой есть и отрицательное значение — минимальное отрицательное значение и максимальное отрицательное значение. Чтобы их получить, достаточно у этих чисел поставить знак «-». У числа `double`, кстати говоря, есть ещё и бесконечности — отрицательная бесконечность и положительная бесконечность, и если мы попытаемся вывести эти значения в консоль, то мы получим `-infinity` и просто `infinity`.

Или, например, минимальное и максимальное значения целого числа, числа класса `integer`.

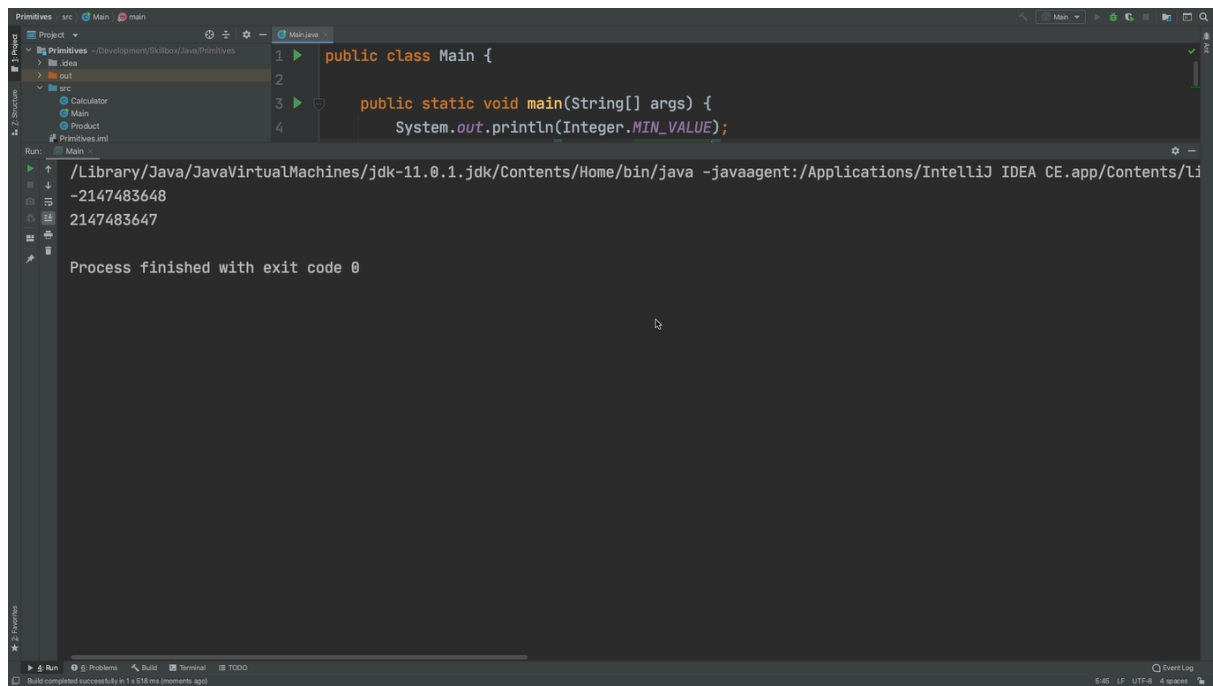


```
1 public class Main {
2
3     public static void main(String[] args) {
4         System.out.println(Integer.MIN_VALUE);
5         System.out.println(Integer.MAX_VALUE);
6     }
7 }
```

The screenshot shows an IDE window titled 'Main.java' with the following code:

```
1 public class Main {
2
3     public static void main(String[] args) {
4         System.out.println(Integer.MIN_VALUE);
5         System.out.println(Integer.MAX_VALUE);
6     }
7 }
```

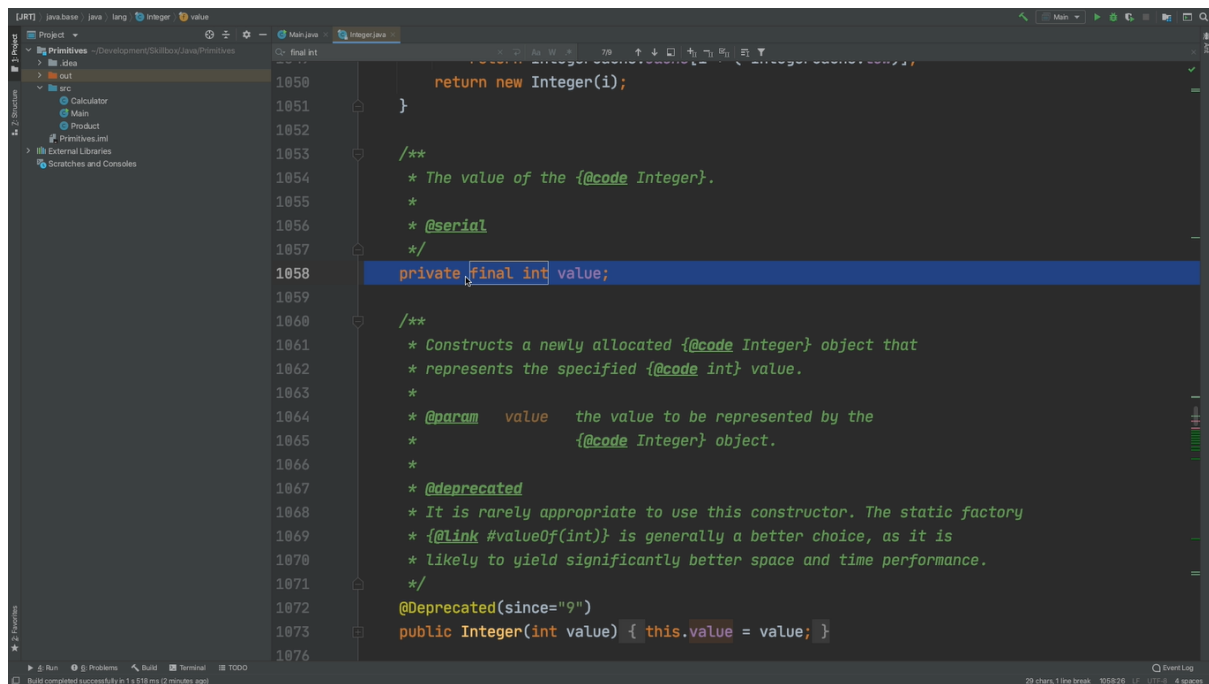
The IDE interface includes a project explorer on the left showing a project named 'Primitives' with subfolders 'src' and 'out'. The 'src' folder contains files 'Calculator.java', 'Main.java', and 'Product.java'. The 'out' folder contains 'Primitives.jar'. The bottom status bar indicates 'Build completed successfully in 1 s 540 ms (a minute ago)'.



Здесь мы просто видим минимальное и максимальное значения.

**02:36–05:28**

Важно понимать, что в классах-обёртках нет никакой магии — это просто классы, которые содержат соответствующие примитивы. Давайте посмотрим исходный код класса `integer`, перейдём внутрь и с помощью поиска найдём слово `final`.



`private final int value` — это то самое значение примитива, которое содержится в объектах этого класса. Именно поэтому эти классы называются классами-обёртками, поскольку они как бы оборачивают значение, представленное примитивом. Поскольку переменная-примитив помечена ключевым словом `final`, она является неизменяемой, и, соответственно, объекты классов-обёрток тоже являются неизменяемыми, то есть иммутабельными.

Давайте вспомним, как работает иммутабельность на простом примере.

У вас есть некая переменная класса `integer`, например возраст некоего Васи, и вы создаёте новую переменную с возрастом некоего Миши и приравниваете её к предыдущей переменной.

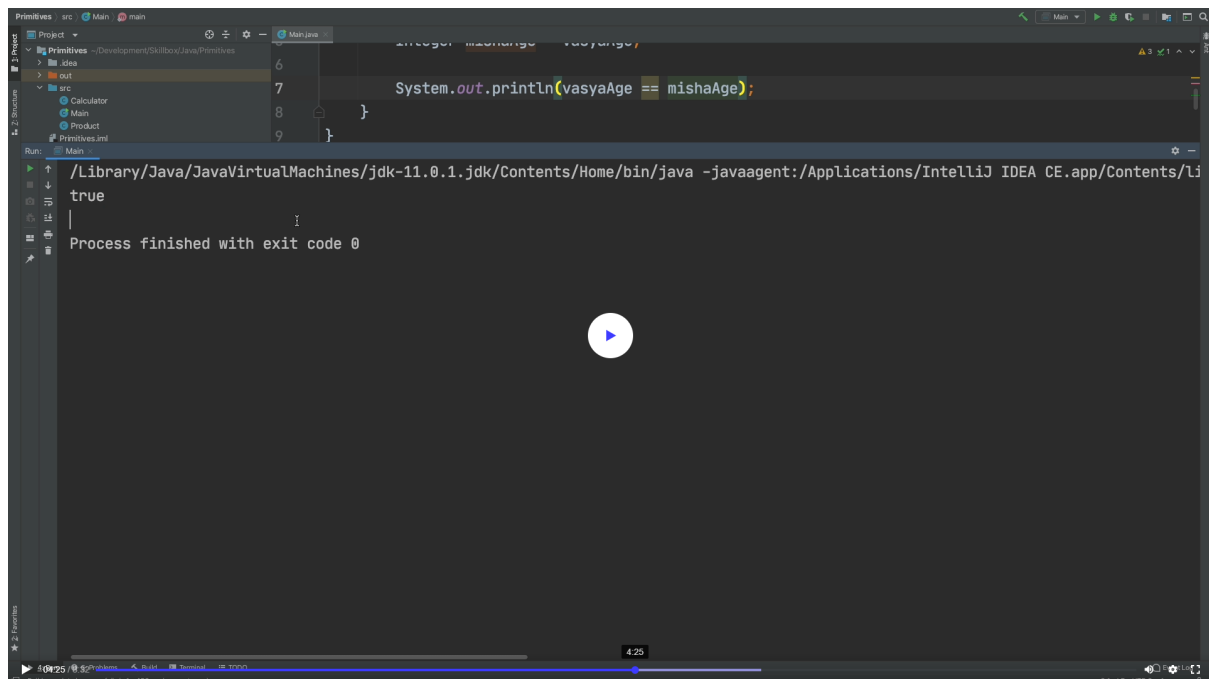
```
public static void main(String[] args) {  
    Integer vasyaAge = 6;  
    Integer mishaAge = vasyaAge ;  
}
```

Мы помним, что переменные-объекты — это ссылки, и когда мы производим присваивание таким образом, то получаем две ссылки

на один и тот же объект в памяти. Мы можем это проверить, используя сравнение в виде двойного равно. Оно как раз будет сравнивать эти самые ссылки.

```
System.out.println(vasyaAge == mishaAge);
```

В результате мы получаем true, то есть обе переменные ссылаются на один и тот же объект.



Но поскольку integer — иммутабельный класс, его изменение приведёт к появлению нового объекта. Если мы здесь напишем новой строкой `vasyaAge = 7`, возникнет новый объект из числа 7 класса integer, и `vasyaAge` и `mishaAge` уже будут разными объектами.

Давай здесь распечатаем значение объекта `mishaAge` и выведем информацию о том, насколько они равные.

`mishaAge` остался равным 6, а `vasyaAge` — это уже новый объект, и мы видим, что ссылки на них разные.

И последнее, о чём стоит помнить, — это то, что объекты классов-обёрток занимают в памяти больше места, чем примитивы,

поскольку они содержат в себе не только переменную-примитив, но и на них ещё есть ссылки, как и у любых других объектов.

**05:28–06:26**

## **Boxing**

Теперь разберём очень важную тему — так называемый boxing (в переводе на русский означает «упаковка» или «оборачивать»). Это автоматическое преобразование примитива в объект соответствующего ему класса-обёртки. Вы только что видели, что мы переменным класса `integer` присваивали простые числа — 6 и 7. Это простейший пример боксинга, то есть мы переменной класса `integer` присваиваем обычный примитив, целое число. Дальше это число автоматически преобразуется в объект класса `integer`, то есть упаковывается в него. Если мы присваиваем уже созданному объекту новое число, то, помимо упаковки числа 7, объекта класса `integer`, происходит присваивание этой переменной ссылки на новый объект.

Ещё раз: boxing — это автоматическая упаковка примитива в объект соответствующего ему класса-обёртки.

## **Boxing**

Автоматическое преобразование примитива в объект соответствующего ему класса-обёртки.

```
Double value = 6.0;
```

**06:26–08:16**

## Unboxing

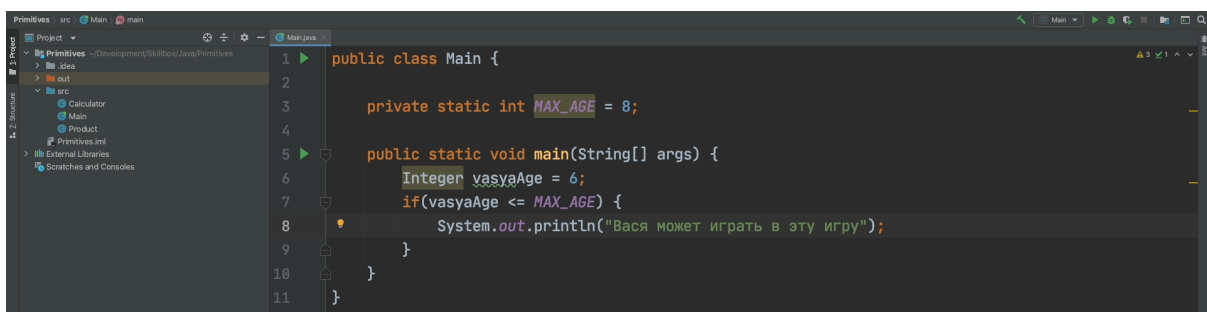
Обратное преобразование называется Unboxing. Вот простейший пример случая, где нужно преобразовать объект класса double в примитив.

# Unboxing

Автоматическое преобразование объекта в примитив, соответствующий классу этого объекта.

```
Double value = 6.0;  
double sum = sum + value;
```

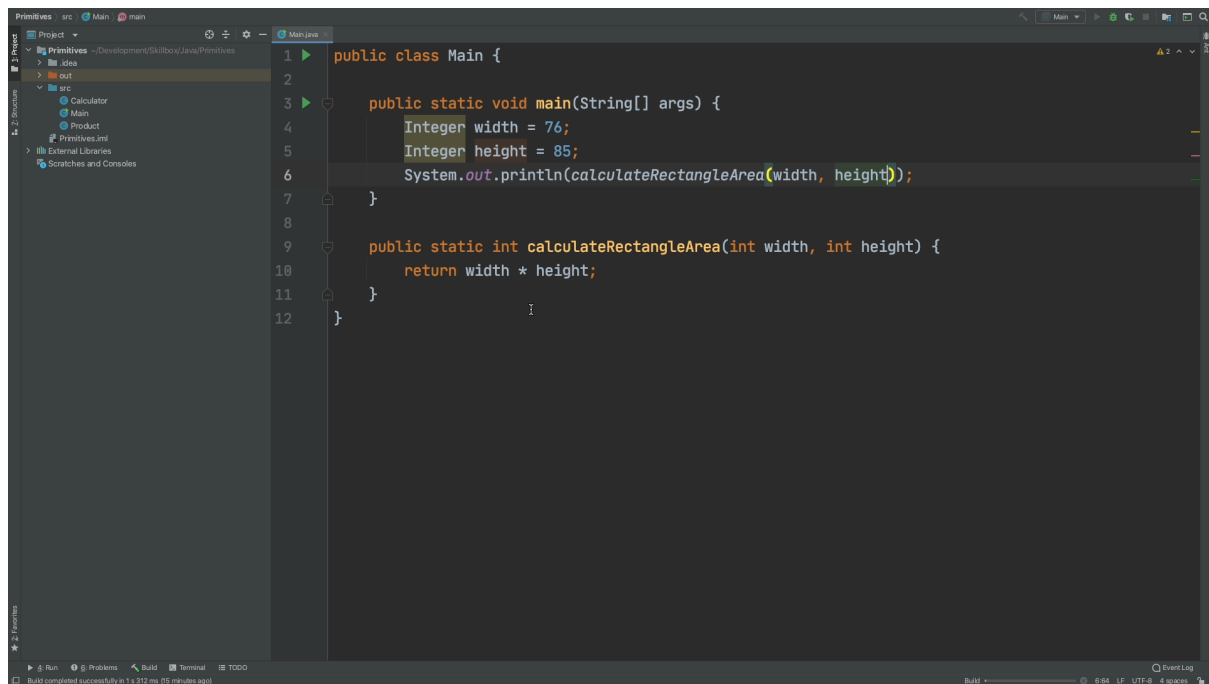
Или, например, если вы будете сравнивать числа и напишите такое сравнение...



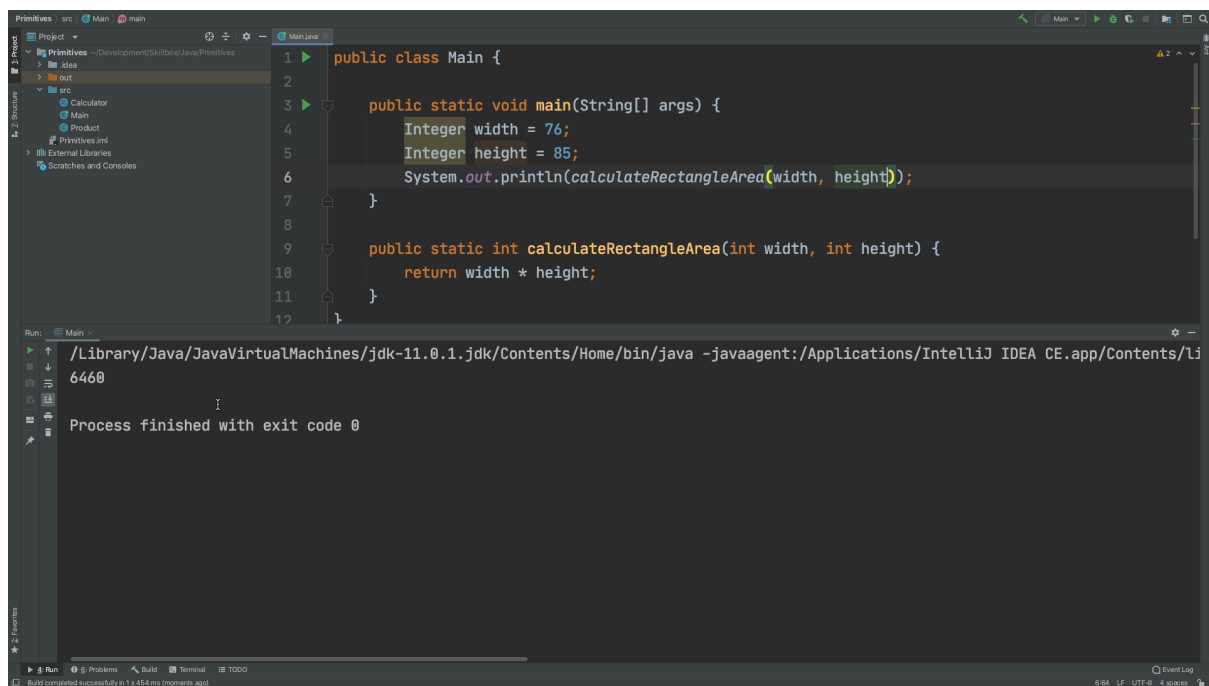
...то в этом случае переменная-объект vasyaAge в выражении сравнения будет автоматически распакована до примитива.

Давайте посмотрим на ещё более сложный пример. Представьте, что вас есть метод, у которого есть параметры типа in. Например, это метод расчёта площади прямоугольника, и сюда передаётся число int (ширина) и число int (высота) Далее в этот метод вы можете передавать как переменные типа int, так и переменные-объекты класса integer.





Запускаем и видим, что всё в порядке.



08:16–08:32

## Итоги

Итак, в этом видео вы познакомились с классами-обёртками, которые в Java существуют для всех примитивов, а также с механизмами их преобразования в примитивны и обратно —

анбоксингом и боксингом.