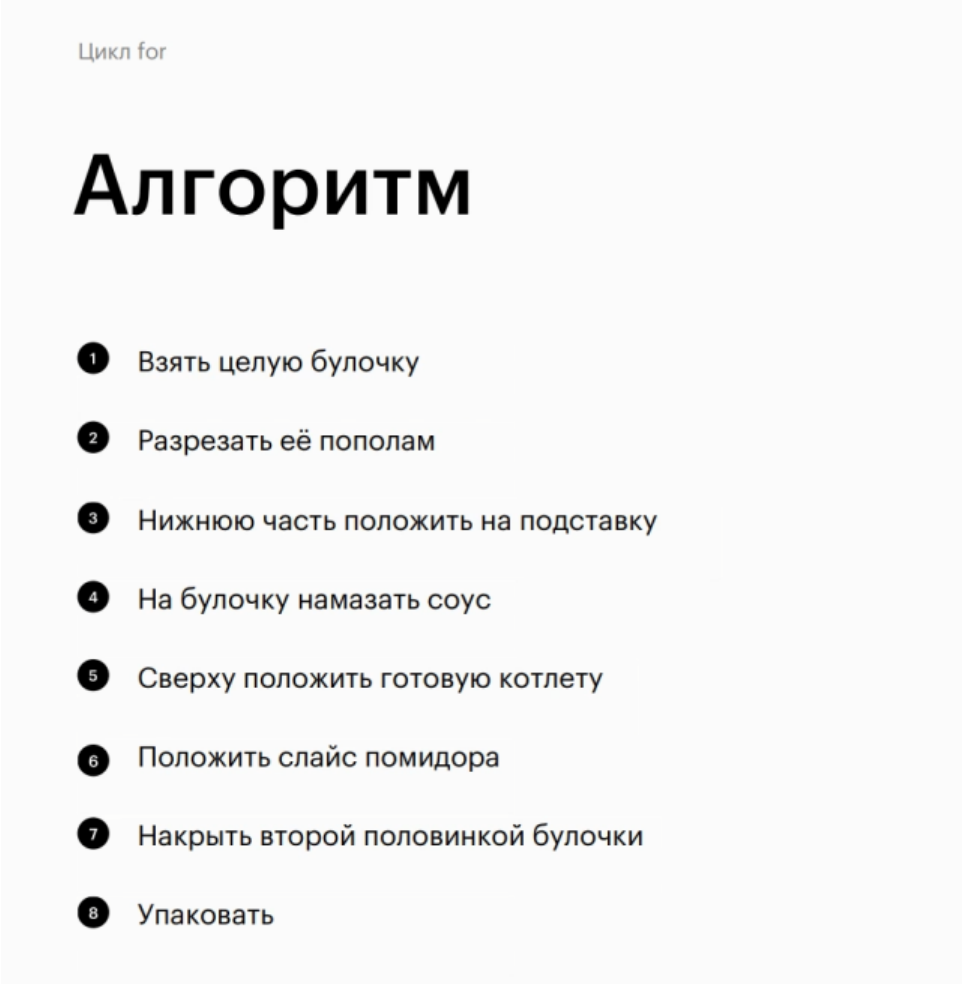
3.2

Цикл for

00:00—03:37
Привет! В этом видеоматериале вы научитесь писать циклы — специальные операторы, которые позволяют повторять определённые действия нужное количество раз.
Как это может быть в жизни? Представьте, что вы пишите программу, которая управляет аппаратом по изготовлению бургеров. На входе у этого аппарата есть готовые булочки, готовые котлеты, соус и, например, помидоры. Задача аппарата — взять булочку, намазать её соусом, положить котлету, затем помидор, сверху накрыть второй половиной булочки и упаковать. В работе такого алгоритма может быть много разных нюансов, но в упрощённом виде этот алгоритм будет выглядеть следующим образом:



Представьте, что вы написали код программы по этому алгоритму и запустили аппарат. Аппарат приготовил один бургер и остановился. Очевидно, что этот код нужно повторить много раз, чтобы аппарат приготовил много бургеров. При этом вы знаете, что у вас котлет обычно всегда меньше, чем загруженных в аппарат булочек, соуса и помидоров, и каждый день в аппарат загружают 500 котлет. Чтобы аппарат изготавливал необходимое количество бургеров, вы просто продублировали свой код 500 раз и в один из дней убедились, что всё работает. И тут ночью вам звонит владелец аппарата и страшно возмущается — оказалось, что последние 300 бургеров упакованы без булочек, там только намазанная соусом котлета и помидор. А ещё весь аппарат в соусе, последние 180 упаковок тоже измазаны соусом и в таком виде отправлены покупателям. Покупатели требуют возврата средств.

В чём была ошибка алгоритма? В том, что код продолжил выполняться, несмотря на то, что один из ингредиентов, а именно булочки, закончился. И что же теперь делать?

Прежде всего, проверять перед изготовлением каждого бургера, что всех ингредиентов достаточно. Но вставлять 500 раз в 500 повторяющихся кодов проверку ингредиентов долго. Есть способ проще.

Код основного алгоритма можно написать один раз, а потом как-то сделать так, чтобы он повторялся сам и повторялся до тех пор, пока выполняются все проверки. Это будет удобно тем, что, в случае необходимости, исправление кода займёт меньше времени — код достаточно будет исправить в одном месте.

04:25—06:20

Повторение одного и того же по кругу называется цикличностью, а прописывание такого алгоритма в коде — созданием цикла. Для того, чтобы наш алгоритм повторялся, нужно создать в коде цикл. Для этого давайте сначала познакомимся с теорией.

В Java существует несколько операторов циклов. Вы уже знаете, что такое оператор — это специальные команды, позволяющие выполнять определённые действия. И операторы циклов — операторы, позволяющие выполнять определённый код заданное количество раз.

Один из таких операторов — оператор for. Почему именно for? Ведь слово for в переводе с английского означает «для». На самом деле, у этого названия есть некоторая предыстория. В первых языках программирования цикл называли forward loop. Forward переводится с английского как «вперёд», а loop — «петля» или «цикл». Для удобства название сократили до forloop, а затем и вовсе до for. И теперь, в большинстве языков программирования, используется именно слово for. Оператор for, пожалуй, чаще всего используется для реализации циклов.

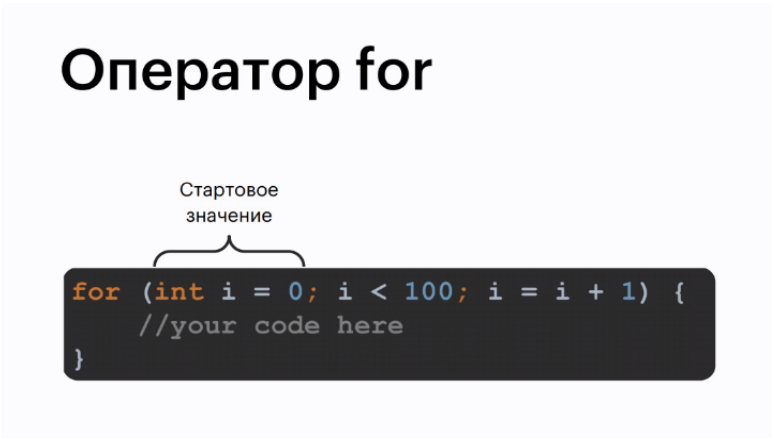
Давайте посмотрим, как он пишется в коде. В коде оператор for пишется следующим образом: сначала пишется сам оператор — слово for, затем круглые скобки, в которых указываются различные параметры этого оператора, и затем пишутся фигурные скобки. В фигурных скобках вы можете написать любой код, и он будет выполняться в зависимости от параметров цикла — он может не выполниться ни одного раза, выполняться один или несколько раз или может выполняться бесконечно, то есть до тех пор, пока программа работает. Давайте теперь подробно разберём параметры оператора for, которые пишутся в круглых скобках.

```
for (int i = 0; i < 100; i = i + 1) {
    //your code here
}
```

06:21—09:00

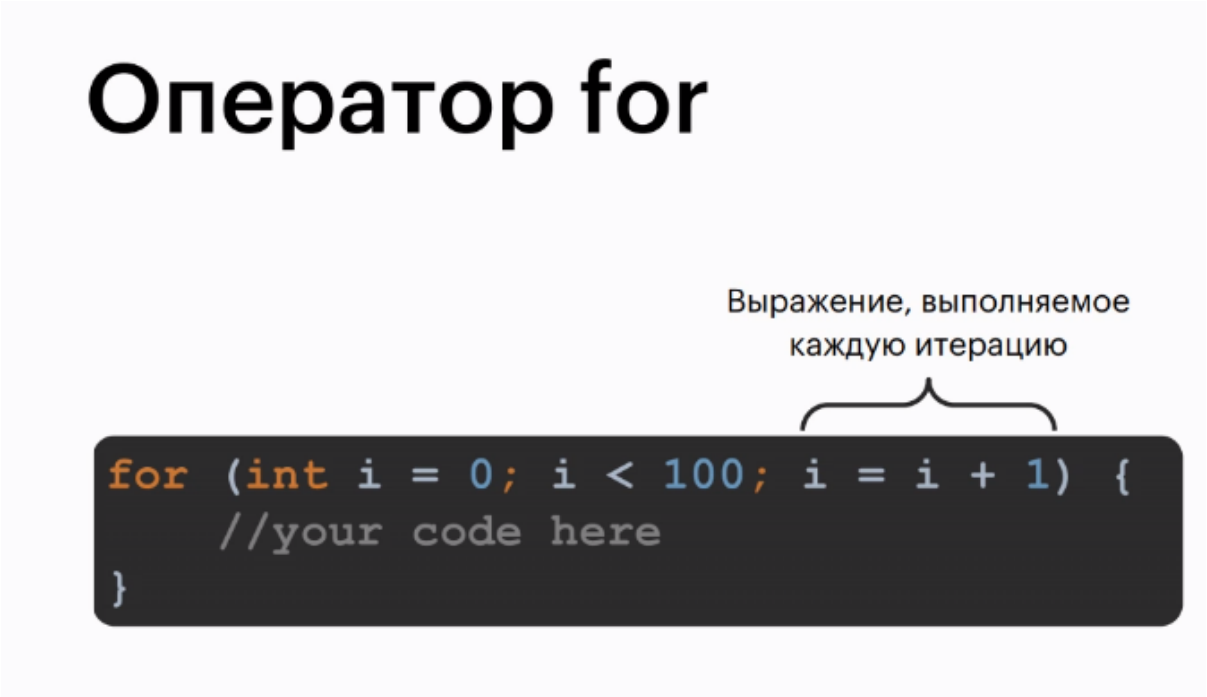
Этих параметров три, и они разделены точкой с запятой (;), после которой принято ставить пробел.

Первый параметр — это так называемое стартовое значение.



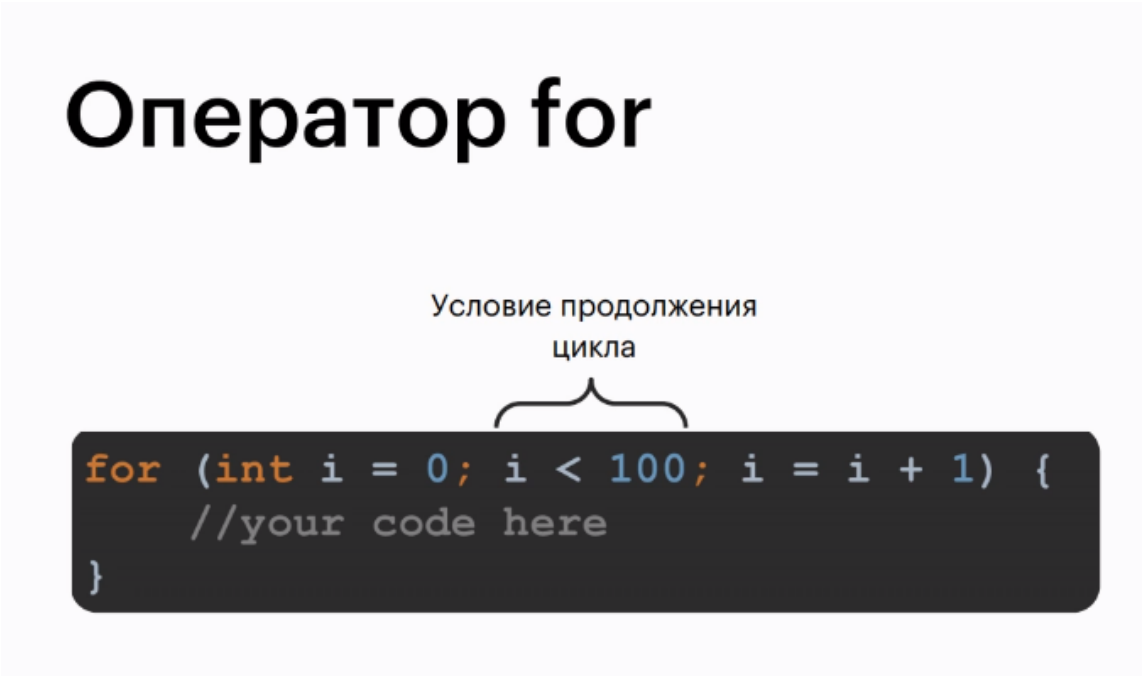
Обычно это некое целое число, которое будет меняться в процессе выполнения цикла. Изначально, когда цикл только начнёт выполняться, это переменная будет равна тому значению, которое здесь задано — в данном случае, нулю. Эту переменную принято называть буквой i от слова iteration, т. е. итерация. Итерация — это один шаг чего-либо, что повторяется. В нашем случае цикл будет повторять или повторно выполнять какой-то код. Такое повторение — это одна итерация цикла.

Итак, первый параметр — это стартовые значение. Ко второму параметру мы сейчас вернёмся, а третий параметр — некое выражение, которое, как и код в фигурных скобках, будет выполняться каждую итерацию цикла.



В примере написано выражение `i = i + 1`, т. е. каждую итерацию

цикла число i будет становиться больше на единицу.
Теперь вернёмся ко второму параметру. Второй параметр — это условие продолжения цикла.

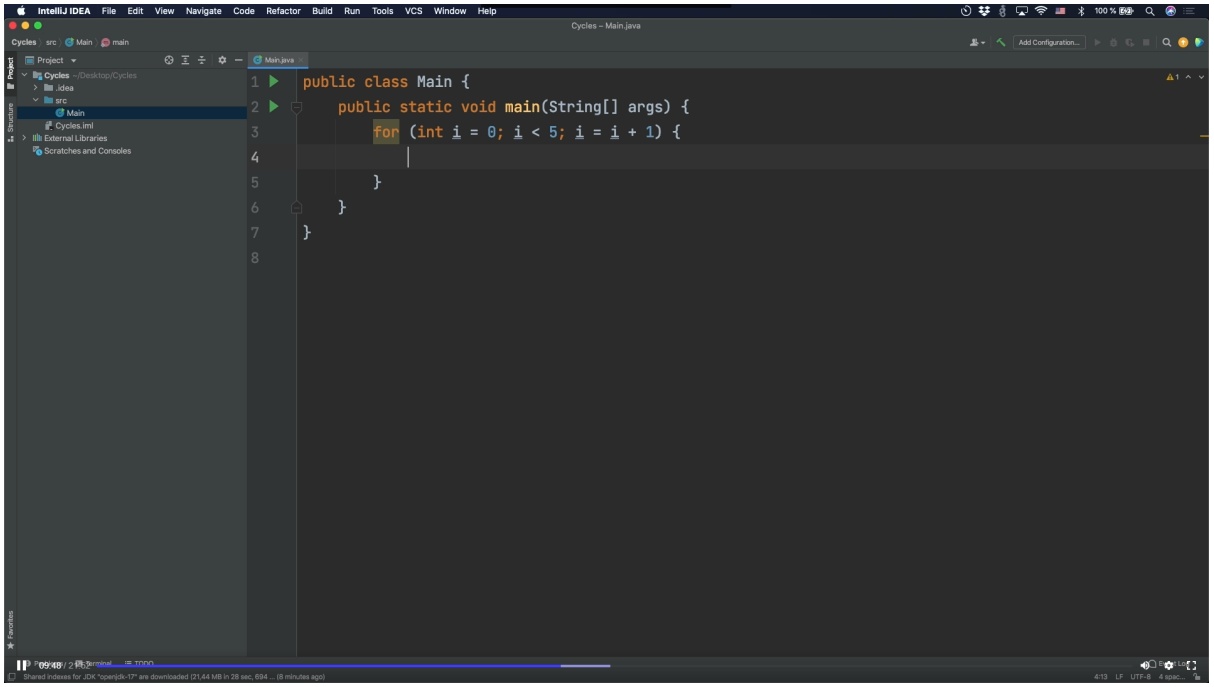


Это выражение, в результате выполнения которого будет появляться значение типа boolean — либо true, либо false. Это выражение будет проверяться каждую итерацию цикла и будет помогать программе, т. е. оператору цикла for, понять, когда повторение нужно остановить. В случае с аппаратом, который изготавливает бургеры, — понять, что нужное количество бургеров уже изготовлено и пора остановиться.
В данном примере написано, что цикл будет выполняться до тех пор, пока значение переменной i меньше 100. Как только значение станет равным 100, цикл прекратит своё выполнение.
Давайте ещё раз закрепим.
В круглых скобках задано стартовое значение переменной `i=0`. Каждую итерацию цикла эта переменная будет увеличиваться на единицу — это прописано в выражении в качестве третьего параметра цикла `i = i + 1`. Это будет происходить до тех пор, пока будет выполняться второй параметр цикла выражения `i<100`.

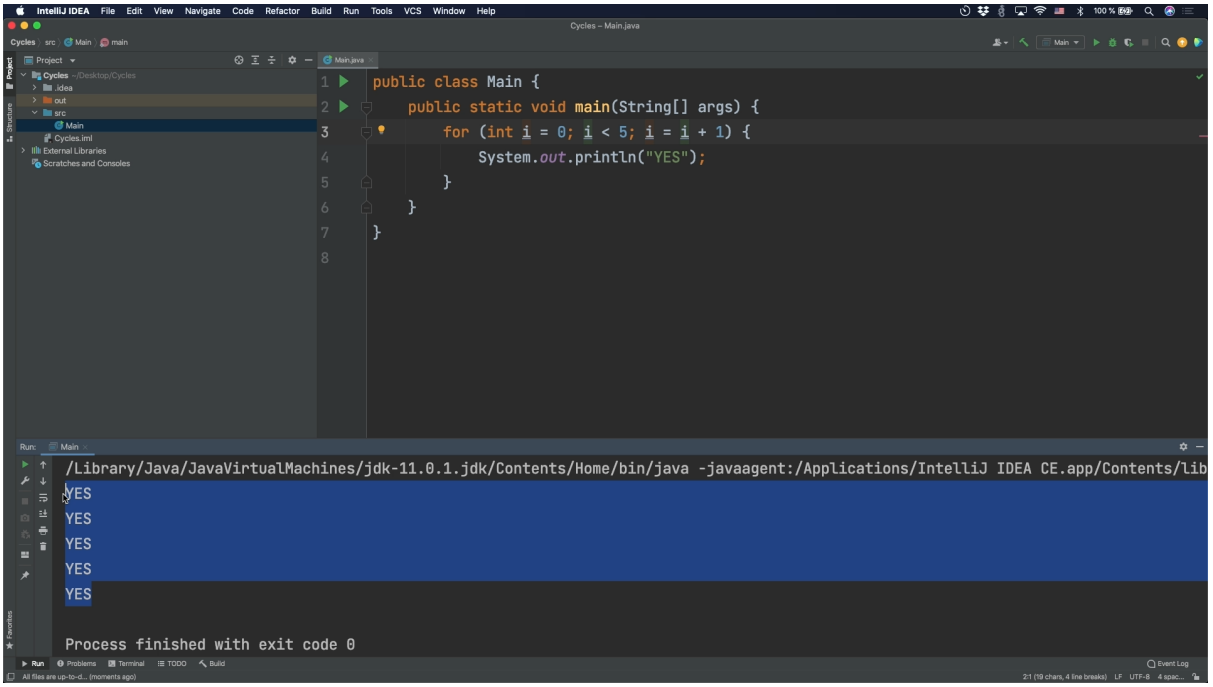
09:00—14:30
Давайте теперь посмотрим на практике, как работает такой цикл.

1. Напишем сначала оператор for.
2. Затем напишем все его параметры:
   2.1. Стартовые значения переменной `i=0` (ставим «;» и пробел).
   2.2. `i<5` — условие продолжения цикла (ставим «;» и пробел).
   2.3. `i = i + 1` — выражение, которое будет выполняться каждую итерацию цикла.

Цикл должен будет выполниться ровно пять раз, поскольку стартовое значение числа i=0, каждую итерацию оно будет увеличиваться на единицу, и это будет происходить до тех пор, пока оно не станет равным 5. Как только оно станет равным 5, цикл перестанет выполняться, и код внутри фигурных скобок тоже не будет выполняться.



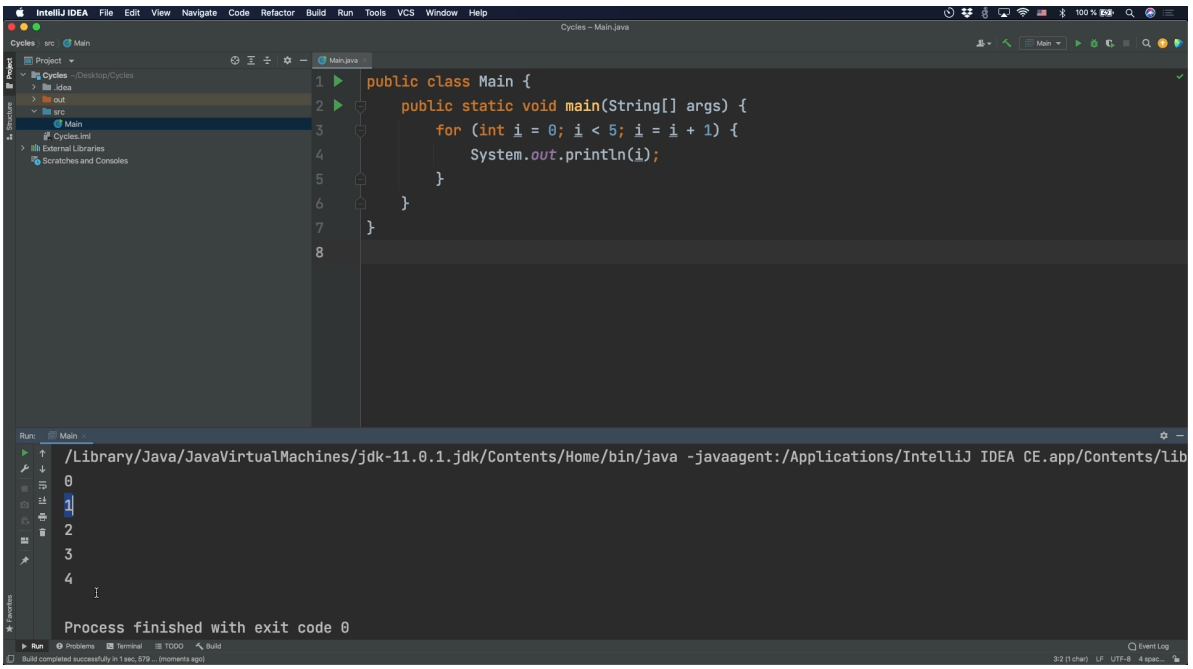
Для того, чтобы наглядно убедиться, что наш цикл работает верно, давайте внутри него напишем вывод в консоль — слово yes — и запустим этот код. Можно увидеть, что цикл действительно выполнил наш код пять раз — слово yes напечатано в консоли пять раз.



Давайте теперь посмотрим, как увеличивается число i каждую итерацию цикла.

Это очень просто сделать — число i доступно в самом цикле, его можно вывести в консоль вместо слова yes — достаточно ввести i, после чего мы точно так же запускаем код.

Можно убедиться в том, что всё работает: число i сначала равно нулю, затем единице — и так до 4. Когда i станет равным 5, цикл прекратит своё выполнение, т. к. перестанет выполняться второй параметр цикла.

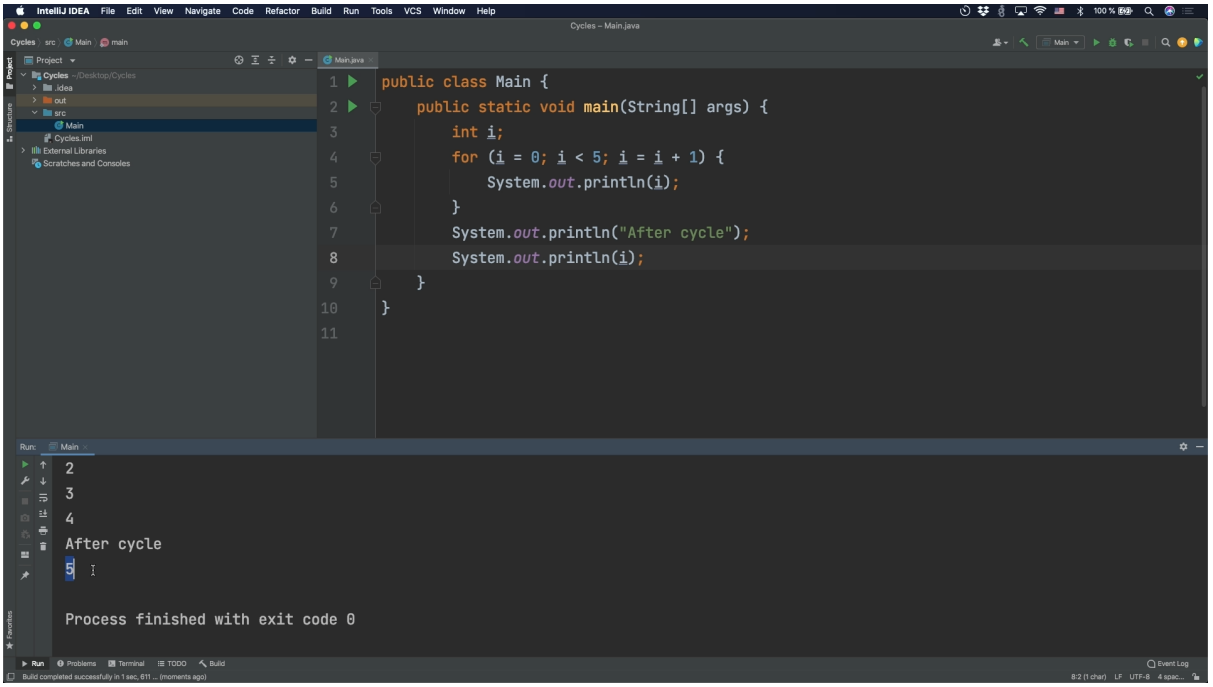


Мы можем убедиться в том, что число i действительно становится равным 5, если выведем его в консоль после цикла.

Давайте сначала выведем фразу After Cycle и затем выведем число i. Можно увидеть, что среда разработки выделила эту переменную красным цветом, потому что число i доступно только внутри цикла, внутри фигурных скобок. Для того, чтобы проверить, чему оно будет равно, его нужно задать перед циклом, то есть написать вот таким образом:

```
int i;
for (i=0; i<5; i = i +1;) {
      System.out.println(i);
}
```

После того, как мы задали число i перед циклом, запускаем код и видим, что код работает — число i стало равно 5, и на этом цикл прекратился.



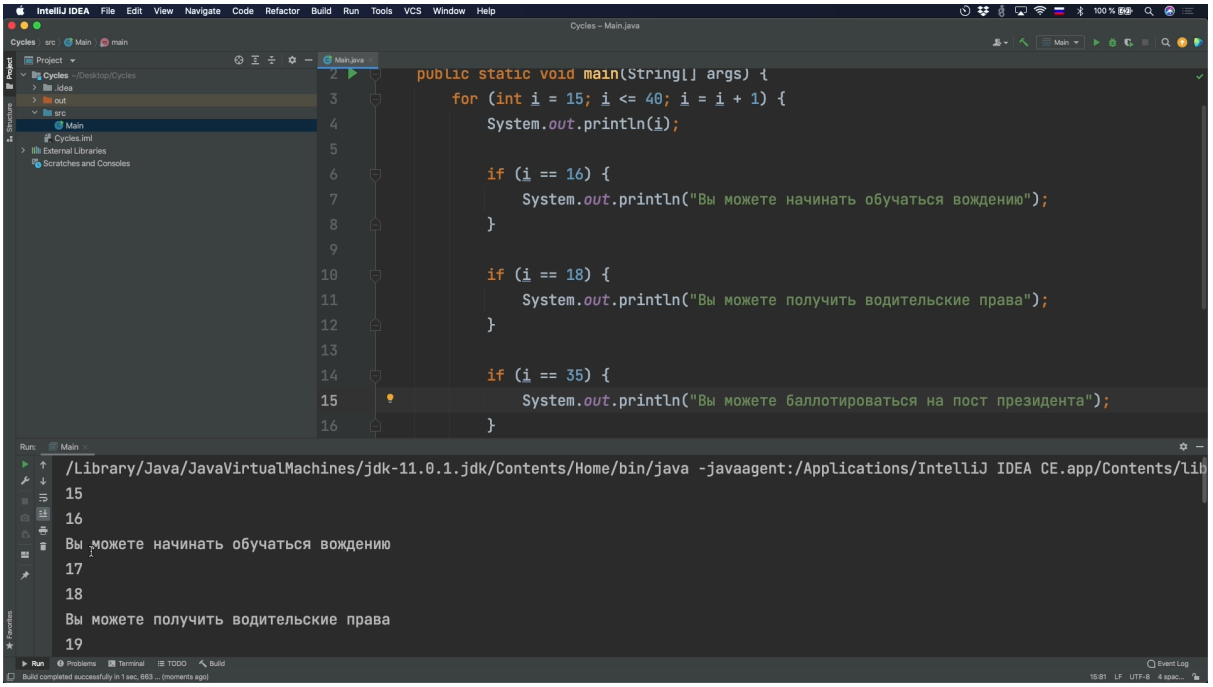
Три параметра внутри круглых скобок можно менять для решения различных задач.

Давайте закрепим понимание и напишем ещё одну программу. Новая программа будет выводить в консоль возраст, например, от 15 до 40 лет, и писать сообщение о том, что можно начинать делать в том или ином возрасте по закону. Например, обучаться вождению можно с 16 лет, получить права и водить — с 18, стать президентом — с 35 лет.

Давайте напишем такой цикл: i будет от 15 до 40 лет, и мы сделаем так, чтобы этот диапазон был до 40 лет включительно, то есть поставим здесь i меньше или равно 40. Внутри цикла напишем более сложный код, чем просто вывод числа, — напишем проверку того, чему это число равно:

- если i=16, то мы выведем в консоль информацию о том, что вы можете начинать обучаться вождению;
- если i=18, то вы можете получить водительские права;
- если i=35, то с этого возраста вы можете баллотироваться на пост президента.

Можно запустить код и убедиться в том, что он работает правильно.



Давайте повторим ещё раз, как мы написали этот код.
Мы написали оператор for, в начале которого переменная i=15, и далее каждую итерацию этого цикла переменная i увеличивается на единицу до тех пор, пока i<=40. Как только она становится равной 41, цикл прекращает своё выполнение. На каждой итерации цикла мы вводим число и сравниваем его с тремя числами: 16, 18 и 35. Далее мы выводим в консоль соответствующие сообщения.
Таким образом, мы научились писать цикл и сможем в дальнейшем его использовать для решения различных задач, в которых требуется что-то повторять или перебирать числа в определённом диапазоне цифр.

14:30—16:45

Циклы также можно использовать для обратного отсчёта, например, вам нужно получить числа от 50 до 1 именно в обратном порядке. Тогда мы можем написать:

```
for (int i=50; i>0; i = i - 1)
```

После этого выведем число i в консоль:

```
System.out.println(i);
```

После этих шагов можно запустить код и убедиться в том, что всё работает правильно. Если бы мы умели приостанавливать выполнение программы на какое-то время, например, на секунду, то можно было бы создать таймер обратного отсчёта, который бы раз в секунду печатал число на единицу меньше 50: 49, 48 и так далее. Но мы пока этого не умеем, поэтому просто выводим числа в обратном порядке.

Мы также можем произвольно менять третий параметр оператора. Например, мы можем начать с его помощью выводить только чётные числа, прибавляя каждый раз не единицу, а двойку:

```
for (int i=0; i<=100; i = i + 2)
```

Тогда, после запуска программы, в консоли будут выведены только чётные числа. Но здесь важно, чтобы стартовое значение тоже было чётным или равным нулю. Если стартовое число будет равно, например, единице, то вы получите нечётные числа.

16:45—20:48

На самом деле, каждый из параметров цикла не является обязательным — мы можем не указывать эти параметры, как один, так и несколько. Если не указать сразу все три параметра, то мы получим бесконечный цикл, который можно будет завершить только завершением программы. Таким образом работал аппарат по изготовлению бургеров: он продолжал делать бургеры, несмотря на то, что булочки закончились.

Однако, бесконечные циклы всё же иногда могут быть нужны. Например, вы пишите какое-нибудь консольное приложение, которое задаёт пользователю вопрос, пользователь на него отвечает, программа что-то делает с его ответом и снова задаёт ему тот же вопрос.

Пусть для примера у нас будет приложение, которое умножает одно число на другое.

Поскольку нам нужен оператор для бесконечного цикла, мы должны здесь написать два раза «;», то есть разделитель параметров:

```
for (; ;) {
System.out.println ("Введите первое число: ");
int value1 = new Scanner(System.in). nextInt();
System.out.println ("Введите второе число: ");
int value2 = new Scanner(System.in). nextInt();
int result =  value1 *  value2;
System.out.println  ("  Произведение  чисел  равно  :  "  +
result);
}
```

После этого запустить код, ввести первое число, потом второе, и получить их произведение. Как только мы получим результат, программа снова попросит нас ввести первое число, поскольку цикл бесконечный. Прервать выполнение такой программы можно только завершив её.

В цикле можно указывать или не указывать любое количество параметров. В предыдущем примере мы не указали сразу три параметра, но можно написать цикл и не указать какой-то один параметр, например, первый.

Выглядеть это будет следующим образом:

```
int i = 0
for (; i < 100; i = i + 3) {
System.out.println(i);
}
```

При запуске кода вы сможете убедиться, что он работает верно — на консоли будут отображаться числа от 0 до 99 с разницей в 3.

Если убрать из кода второй параметр, то цикл станет бесконечным.

```
for (i = 0; i = i + 3) {
System.out.println(i);
}
```

В результате, при запуске кода число будет увеличиваться от 0 до бесконечности с разницей в три, но цикл не остановится, поскольку

второй параметр не указан.

Если убрать третий параметр, то с числом i ничего не будет происходить, и так цикл тоже можно сделать бесконечным.

```
for (i = 0; i < 100;) {
System.out.println(i);
}
```

i всё время будет равно 0 и оно никогда не достигнет числа 100.

Мы можем при этом прописывать любые операции с числом i внутри цикла, то есть внутри фигурных скобок.

```
for (i = 0; i < 100;) {
System.out.println(i);
i = i + 2;
}
```

Он полностью будет аналогичен коду, в котором это выражение находится в качестве третьего параметра.
Таким образом, оператор for можно использовать в разных вариациях, меняя его параметры в зависимости от решаемой задачи и необходимости.

Итак, в этом видеоматериале вы познакомились с оператором for, который позволяет создавать различные цифры и перебирать числа в любом диапазоне в прямом и обратном порядке и с любым шагом. В Java, как и во многих других современных языках программирования, циклы можно реализовывать не только с помощью оператора for, но и с помощью операторов while и do while, с которыми мы познакомимся в следующем видеоматериале.