



План занятия

1. [Цикл for: что есть в цикле](#)
2. [Цикл for: движение в обратном направлении и изменение шага](#)
3. [Циклы с условием и вложенные циклы](#)
4. [Цикл while](#)
5. [Бесконечный цикл](#)
6. [Прокачиваем финансовое приложение и добавляем в него циклы](#)



Цикл for: что есть что в цикле

Цикл for: что есть что в цикле

Отвлечёмся от повседневных дел и продолжим погружаться в азы Java. В этой теме вас ждёт знакомство с **циклами**. Они позволяют выполнять одно и то же действие несколько раз. К примеру, представим, вы завели хомяка. Грызуна зовут Байт, он любит есть морковки. Каждый день Байт съедает три морковки. Отобразить это в коде можно так:

```
System.out.println("Байт съел морковку.");  
System.out.println("Байт съел морковку.");  
System.out.println("Байт съел морковку.");
```



Цикл for: что есть что в цикле

Отлично, всё работает, но что если Байт начнёт съедать в день по 10 морковок? Можно повторить строку кода нужное число раз, но тогда программа станет громоздкой. Разработчики вообще не любят дублировать код — это скучно и неудобно. Есть даже профессиональное правило **DRY — Don't Repeat Yourself** (англ. «не повторяй себя»). Поэтому в Java (и в других языках тоже) используются циклы — с их помощью действие выполняется нужное количество раз, но код при этом не дублируется.



Цикл `for`: что есть что в цикле

Существует несколько типов циклов — они начинаются с разных служебных слов и применяются для разных задач. Сначала мы подробно разберём цикл, который начинается со служебного слова `for`. Он используется тогда, когда число повторений известно заранее, — к примеру, сейчас мы знаем, что Байту нужно 10 морковок.



Цикл `for`: что есть что в цикле

Когда вы определили число **итераций** — повторений, которые должен выполнить цикл, нужно его настроить. После слова `for` в круглых скобках требуется задать стартовое значение цикла, условие его работы и порядок выполнения этого условия. От этих трёх параметров зависит, откуда цикл начнёт свое движение, сколько раз выполнится и как будет меняться его начальное значение. После этого в фигурных скобках указывается тот участок кода, который нужно повторять, — он называется **телом цикла**.

Цикл for: что есть что в цикле

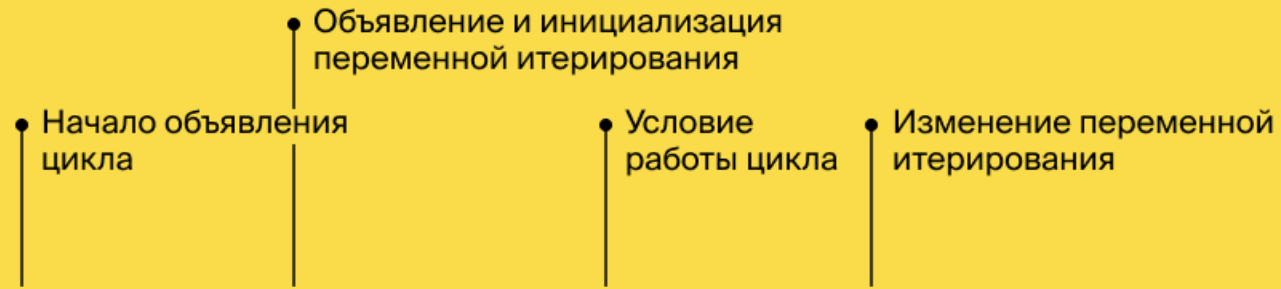
Только после того, как цикл выполнит все итерации, заложенные в его условии, программа начнёт выполнять код, идущий следом. Всё вместе это выглядит так:

```
for (int i = 1; i <= 10; i = i + 1) { // Объявление цикла и его настройки
    System.out.println("Байт съел морковку."); // Тело цикла, выполняется при каждой
итерации
}
// Цикл завершён, выполнится эта строка:
System.out.println("Все морковки съедены. Морковок больше нет.");
```

Цикл for: что есть что в цикле

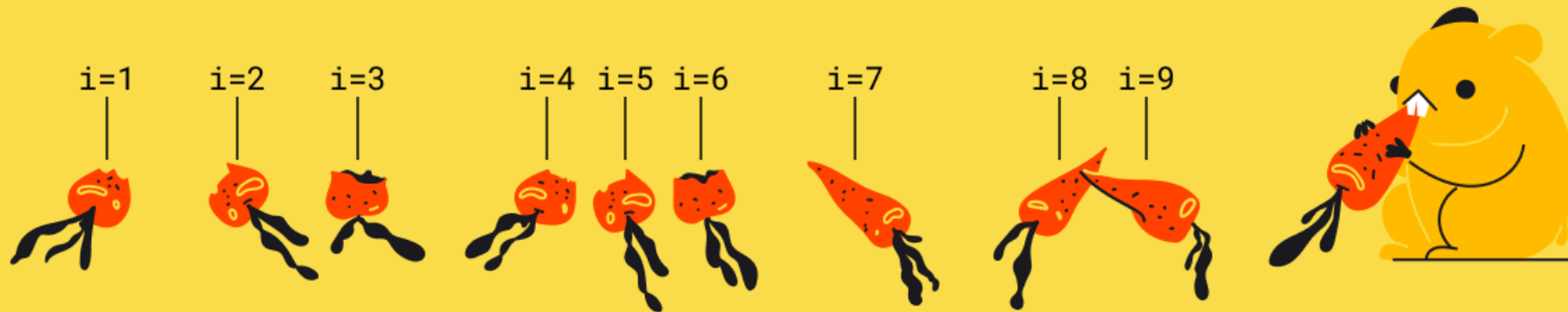
Сначала мы объявили **переменную итерирования**: с помощью неё настраивается количество повторов в цикле. Имя переменной вы выбираете сами, но традиционно её называют `i` — как и в нашем примере. Начальное значение равно единице (`int i = 1`) — именно с этой цифры ведётся отсчёт цикла.

Далее идёт условие, при котором цикл будет работать. Так как Бит съест 10 морковок, условие такое: `i <= 10`. Пока оно истинно — цикл выполняется, а значение переменной `i` пошагово изменяется. Когда условие задано, нужно отразить в настройках цикла, что за один подход хомяк съедает одну морковку. Это выглядит так: `i = i + 1` — значение переменной `i` при каждой итерации должно возрастать на единицу.



```
for (int i=1; i<=10; i=i+1) {  
    System.out.println ("Байт съел морковку");  
}
```

• Тело цикла



Цикл for: что есть что в цикле

В теле цикла укажем текст, который нужно повторять — `System.out.println("Байт съел морковку. ")`; . Когда все итерации цикла будут выполнены, появится сообщение о том, что морковки съедены. Результат запуска кода будет таким:

```
Байт съел морковку.  
Байт съел морковку.  
Байт съел морковку.  
Байт съел морковку.  
Байт съел морковку.  
Байт съел морковку.  
Байт съел морковку.  
Байт съел морковку.  
Байт съел морковку.  
Байт съел морковку.  
Все морковки съедены. Морковок больше нет
```



Для чего используется цикл `for`?

- A) Чтобы повторять код бесконечно.
- B) Чтобы повторить участок кода заданное число раз.
- C) Для того, чтобы программа работала быстрее.
- D) Чтобы накормить хомяка.



Ответ

А) Чтобы повторять код бесконечно.

Нет, бесконечные циклы создаются иначе. Особенность `for` в ограниченном количестве итераций.

В) Чтобы повторить участок кода заданное число раз.

Правильно! Именно для этого цикл `for` и нужен! Количество итераций указывается в условии цикла.

С) Для того, чтобы программа работала быстрее.

Нет, цикл `for` нужен не для этого.

Д) Чтобы накормить хомяка.

Хороший вариант, но нет, циклами сыт не будешь!



Цикл for: что есть что в цикле

Если требуется увеличить или уменьшить количество итераций, нужно изменить условие цикла. К примеру, десять морковок для Байта много — он сильно прибавил в весе. Снизим число морковок до пяти в день. В новой условии нужно отразить, что переменная итерирования `i` должна быть меньше или равна 5.

Цикл for: что есть что в цикле

Для контроля съеденного запишем все морковки. Для этого также можно использовать переменную итерирования — она доступна внутри цикла не только в его настройках, но и в теле (при этом не видна за пределами цикла). В примере с Байтом значение переменной будет меняться при каждой итерации в последовательности от 1 до 5 — если поставить её в тело цикла, то будет учтена каждая морковка. Код получится таким:

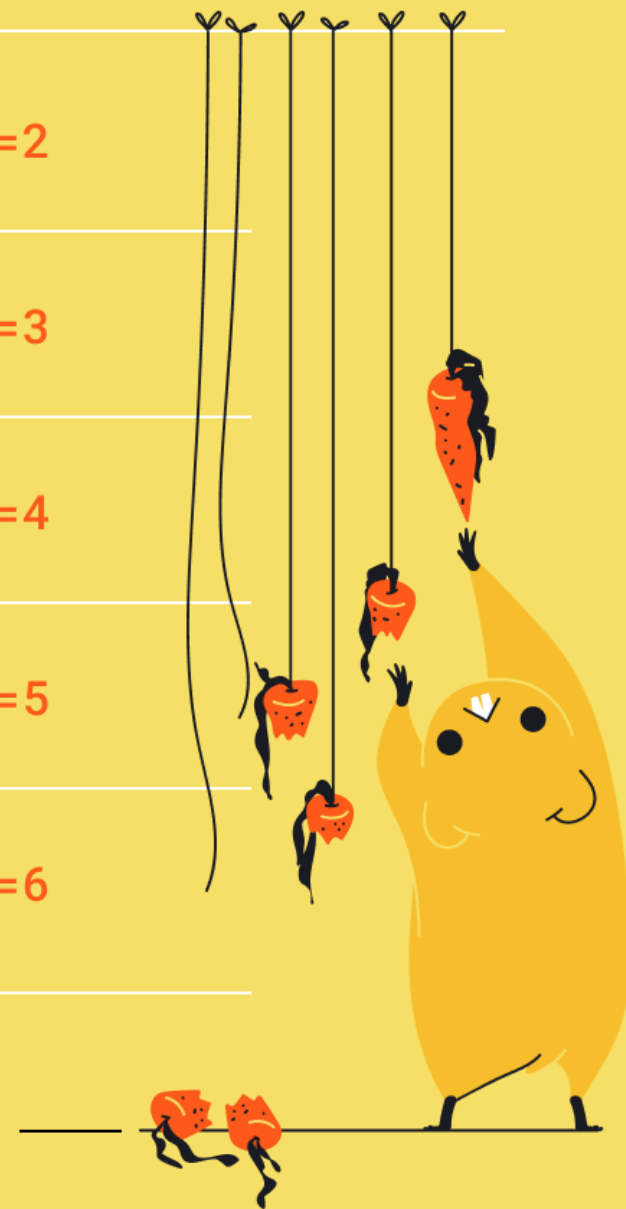
```
public class Practice {  
    public static void main(String[] args) {  
        for (int i = 1; i <= 5; i = i + 1) {  
            System.out.println("Байт съел " + i + "-ю морковку.");  
        }  
    }  
}
```



Цикл for: что есть что в цикле

Во время выполнения цикла происходит следующее: при каждой итерации значение `i` увеличивается на единицу, после чего программа проверяет заданное условие (`i <= 5`) на истинность. Цикл повторяется до тех пор, пока условие истинно. Если Байт захочет съесть 6-ю морковку, Java определит, что 6 больше 5 — условие ложно, и цикл завершится. Измените условие цикла (к примеру, `i <= 3`) и убедитесь, что Байт не сможет съесть больше морковок, чем вы укажете.

	Проверка условия выполнения	Действие	Как меняется
1	$i=1, 1 \leq 5?$ Да, продолжаем цикл	Выводим строку: “Байт съел 1-ю морковку”	$i=1+1=2$
2	$i=2, 2 \leq 5?$ Да, продолжаем цикл	Выводим строку: “Байт съел 2-ю морковку”	$i=2+1=3$
3	$i=3, 3 \leq 5?$ Да, продолжаем цикл	Выводим строку: “Байт съел 3-ю морковку”	$i=3+1=4$
4	$i=4, 4 \leq 5?$ Да, продолжаем цикл	Выводим строку: “Байт съел 4-ю морковку”	$i=4+1=5$
5	$i=5, 5 \leq 5?$ Да, продолжаем цикл	Выводим строку: “Байт съел 5-ю морковку”	$i=5+1=6$
6	$i=6, 6 \leq 5?$ Нет, останавливаем цикл		





Цикл for: что есть что в цикле

Обратите внимание, в примере с Байтом и морковками мы использовали знак нестрогого сравнения: \leq (меньше или равно). Задать условие, что в цикле должно быть пять повторений, можно и по-другому, с помощью знака строгого сравнения: $i < 6$. Выражения $i \leq 5$ и $i < 6$ полностью равнозначны — можно использовать их оба.

Прочитайте код цикла и определите, что произойдёт в результате его выполнения

```
for (int i = 0; i < 8; i = i + 1) {  
    System.out.println("Привет!");  
}
```

- A) 8 раз выведено слово "Привет!"
- B) 7 раз выведено слово "Привет!"
- C) 9 раз выведено слово "Привет!"
- D) Цикл не работает, он настроен неверно — i не может равняться нулю.

Ответ

А) 8 раз выведено слово "Привет!"

Верно! Отсчёт начинается с нуля, а условие выхода из цикла $i < 8$ — это строгое сравнение. Тело цикла выполнится ровно 8 раз.

В) 7 раз выведено слово "Привет!"

Отсчёт начинается с нуля, а условие выхода из цикла $i < 8$ — это строгое сравнение. Этот ответ был бы верным, если бы условие было $i \leq 6$ или $i < 7$.

С) 9 раз выведено слово "Привет!"

Вы ошиблись. Стартовое значение переменной итерирования — ноль ($i = 0$), а условие выхода из цикла $i < 8$. Цикл сработает 8 раз.

Д) Цикл не сработает, он настроен неверно — i не может равняться нулю.

В настройках цикла нет ошибки, мы можем задавать любые значения для переменной i .

Цикл for: что есть что в цикле

Переменная итерирования может принимать не только положительные значения или равняться нулю, но и быть отрицательной. Предположим, вы с Байтом решили переехать в новую квартиру и в вашем доме теперь есть подземная парковка (минус первый этаж) и даже лобби (нулевой этаж). Ваша квартира находится на третьем этаже, и вам нужно подняться в неё на лифте с паркинга. В таком случае запрограммировать табло лифта можно так:

```
for (int i = -1; i <= 3; i = i + 1) {  
    System.out.println("Этаж " + i);  
}
```



Результат

На экране будет напечатано:

```
Этаж -1  
Этаж 0  
Этаж 1  
Этаж 2  
Этаж 3
```

Задача

Запрограммируйте с помощью цикла **for** подъём лифта в доме, где 11 этажей и просторная трёхуровневая парковка. В результате запуска цикла должен отображаться каждый этаж с минус третьего по одиннадцатого.

```
public class Practice {  
    public static void main(String[] args) {  
        // Вместо точек подставьте нужные выражения  
        for (...; ...; ...) {  
            System.out.println("Этаж " + ...);  
        }  
    }  
}
```

Решение

```
public class Practice {  
    public static void main(String[] args) {  
        for (int i = -3; i <= 11; i = i + 1) {  
            System.out.println("Этаж " + i);  
        }  
    }  
}
```


Задача

Циклы — сложная тема. Новой информации оказалось так много, что у вас никак не получается заснуть. Требуется применить проверенный метод — посчитать овец. Напишите цикл, в котором они одна за другой будут прыгать через забор. Вам нужно пересчитать 17 овец. Тогда вы сможете отдохнуть с чистой совестью.

```
public class Practice {  
    public static void main(String[] args) {  
        ... // Здесь нужно объявить цикл  
        System.out.println(... + "-я овечка перепрыгнула через забор.");  
  
        // Цикл завершён, далее выполнится эта строка:  
        System.out.println("Хомяки сыты, овцы целы. Можно отдохнуть!");  
    }  
}
```


Решение

```
public class Practice {  
    public static void main(String[] args) {  
        for (int i = 1; i <= 17; i++) {  
            System.out.println(i + "-я овечка перепрыгнула через забор.");  
        }  
  
        // Цикл завершён, далее выполнится эта строка:  
        System.out.println("Хомяки сыты, овцы целы. Можно отдохнуть!");  
    }  
}
```



**Цикл for: движение в обратном
направлении и изменение шага**

Цикл for: движение в обратном направлении и изменение шага

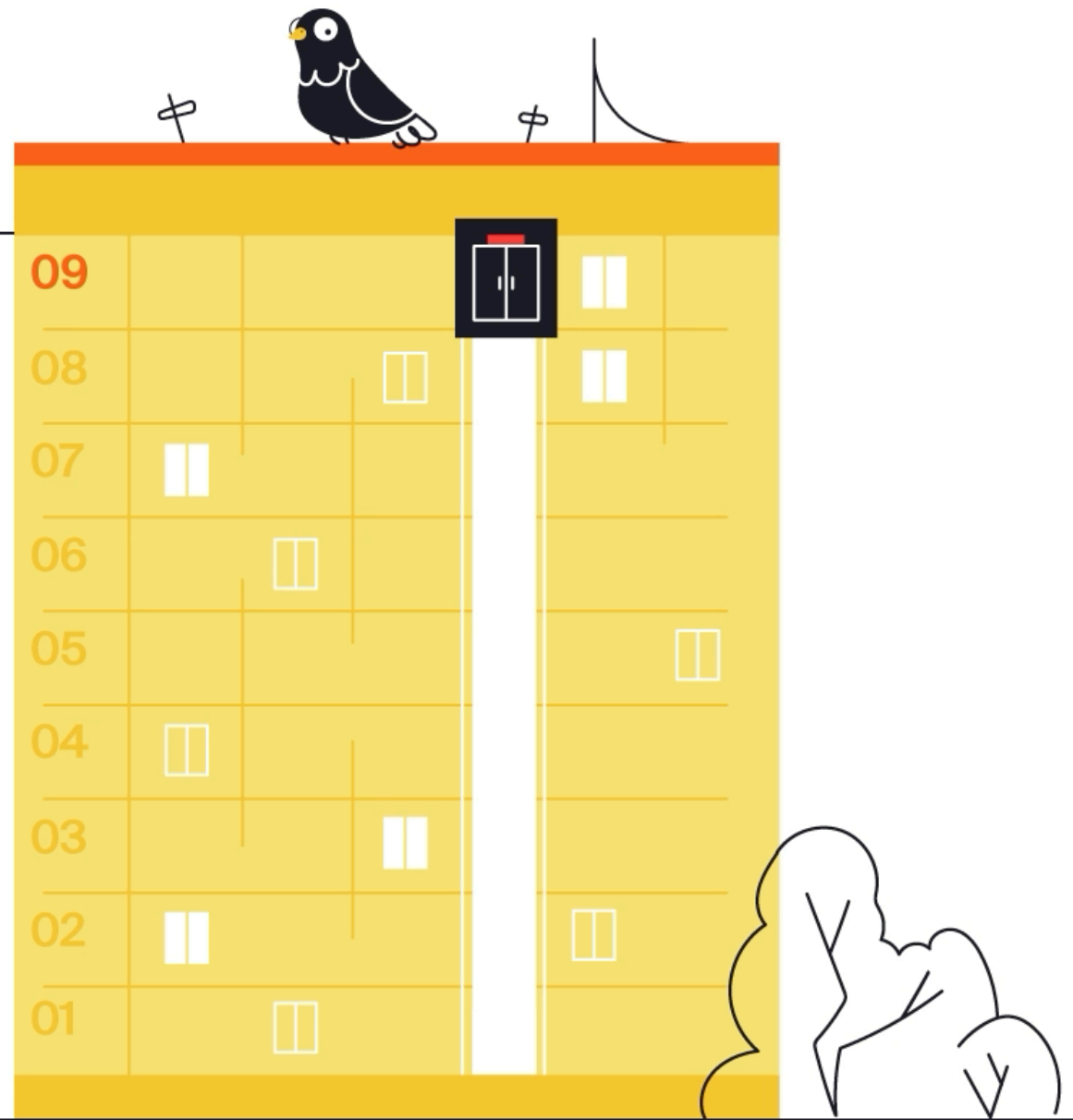
Продолжаем. Весь прошлый урок вы работали с циклами, где переменная итерирования увеличивалась — морковки, овечки и этажи прибавлялись. Теперь представим, что вам нужно запрограммировать цикл **for** в обратном порядке. К примеру, на лифте можно не только подниматься, но и спускаться — номера этажей в таком случае отображаются по убыванию. Код цикла для лифта, который спускается с девятого на первый этаж, получится таким:

```
for (int i = 9; i > 0; i = i - 1) {  
    System.out.println("Этаж " + i);  
}
```

Начальное значение переменной итерирования i равно 9 — лифт начинает движение вниз с девятого этажа. Поскольку цикл должен завершиться, когда лифт доедет до первого этажа, в условии значение переменной i должно быть больше нуля либо больше или равно единице: $i > 0$ или $i \geq 1$. Номера этажей уменьшаются, поэтому при каждой итерации требуется не прибавлять к i единицу, а отнимать — получится $i = i - 1$. Результатом выполнения такого цикла будет:

Этаж	9
Этаж	8
Этаж	7
Этаж	6
Этаж	5
Этаж	4
Этаж	3
Этаж	2
Этаж	1

i=9
лифт находится
на 9 этаже



Цикл for: движение в обратном направлении и изменение шага

Запрограммировать цикл в обратном порядке не сложнее, чем в прямом. Кстати, для уменьшения переменной на единицу $i = i - 1$ можно использовать более простую запись: $i--$ — чем меньше кода, тем легче он читается и воспринимается. Операции $i = i - 1$ и $i--$ равнозначны. Для увеличения переменной i на единицу можно также использовать сокращённое выражение $i++$. На месте i может быть любая другая переменная.

Какие циклы, идут в обратном порядке

- A. `for (int i = 0; i <= 100; i = i + 1)`
- B. `for (int i = 7; i > 1; i--)`
- C. `for (int i = 12; i > 0; i++)`
- D. `for (int k = 3; k > 0 ; k = k - 1)`

Какие циклы, идут в обратном порядке

A. ~~for (int i = 0; i <= 100; i = i + 1)~~

B. for (int i = 7; i > 1; i--)

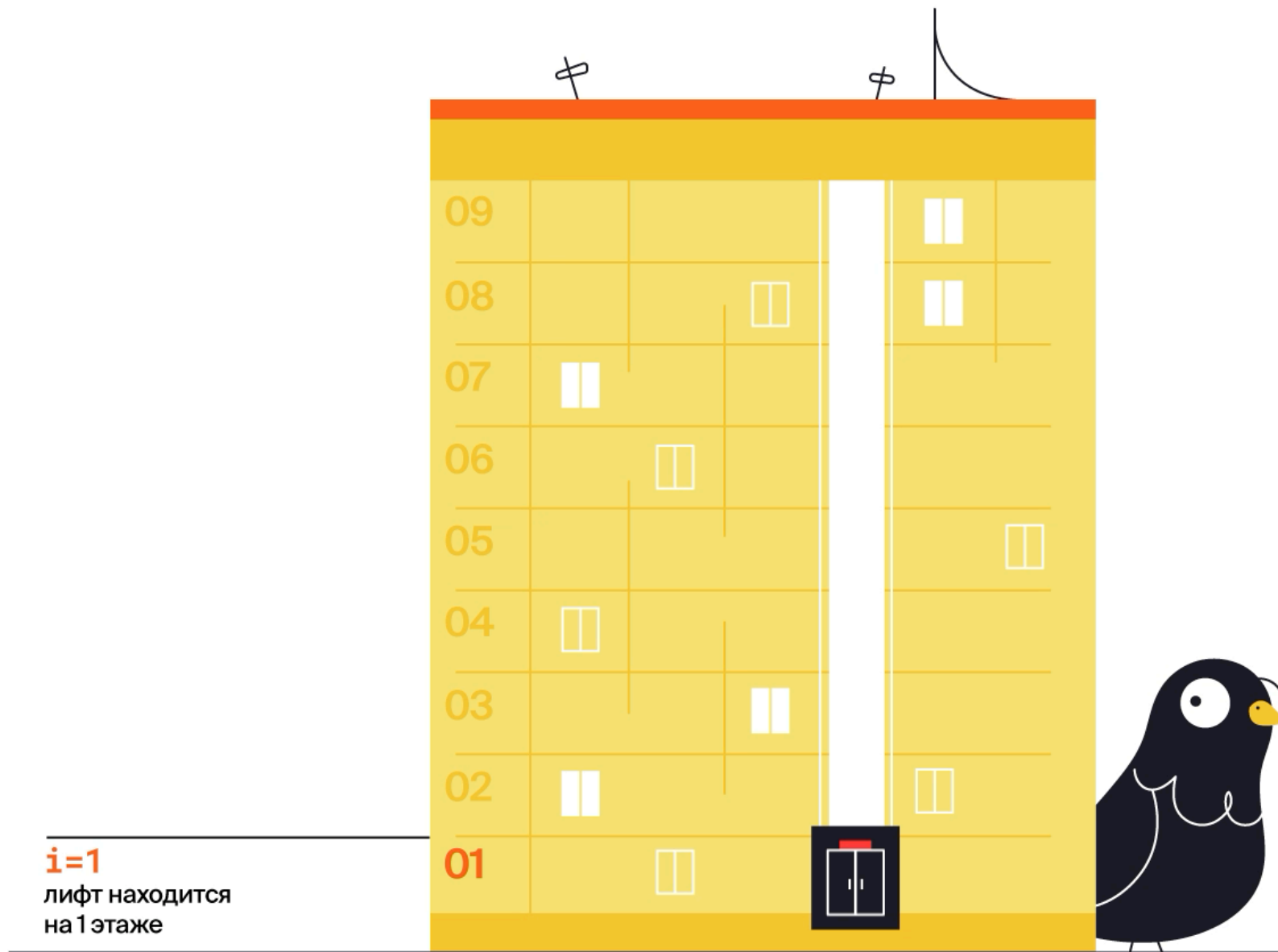
C. ~~for (int i = 12; i > 0; i++)~~

D. for (int k = 3; k > 0 ; k = k - 1)

Переменная итерирования может меняться не только на единицу. Представим, что лифт перепроектировали. Теперь он останавливается только на нечётных этажах. Появилась новая последовательность: вместо 1, 2, 3, 4, 5... стало 1, 3, 5, 7, 9. Чтобы запрограммировать такой лифт, нужно изменить **шаг цикла** — число, на которое после каждой итерации изменяется его переменная. Раньше был шаг 1, а теперь будет 2. Заменяем в коде операцию `i = i + 1` на `i = i + 2`:

```
for (int i = 1; i <= 9 ; i = i + 2) {  
    System.out.println("Этаж " + i);  
}
```

Теперь лифт будет открывать двери только на первом, третьем, пятом, седьмом и девятом этажах. Мы можем сделать и так, чтобы он останавливался через каждые пять этажей ($i = i + 5$) — шаг цикла может быть любым числом. Его можно как увеличивать, так и уменьшать.



i=1
лифт находится
на 1 этаже

Как должен выглядеть цикл, чтобы лифт двигался по чётным этажам сверху вниз?

- A. `for (int i = 9; i >= 1; i = i + 2)`
- B. `for (int i = 8; i >= 1; i = i - 1)`
- C. `for (int i = 8; i >= 1; i = i - 2)`
- D. `for (int i = 8; i >= 1; i = 2)`



Как должен выглядеть цикл, чтобы лифт двигался по чётным этажам сверху вниз?

A. ~~for (int i = 9; i >= 1; i = i + 2)~~

B. ~~for (int i = 8; i >= 1; i = i - 1)~~

C. for (int i = 8; i >= 1; i = i - 2)

D. ~~for (int i = 8; i >= 1; i = 2)~~



Цикл for

Пока проектировали лифты, совсем забыли о Байте, а он тем временем съел всю морковь, и теперь вам нужно идти в магазин. До этого момента мы использовали цикл, чтобы напечатать строку нужное число раз. Однако его также можно использовать для расчётов.

К примеру, вы хотите понимать, сколько морковок нужно покупать для Байта в течение недели с учётом того, что он съедает по пять в день. С помощью цикла можно вывести на экран, сколько овощей съедает грызун в течение этих небольших сроков — от одного до семи дней. Заодно напечатаем общий рацион Байта на неделю.

Вот что получится:

```
public class Practice {  
    public static void main(String[] args) {  
        int carrotCount = 0; // Объявляем переменную для общего числа морковок  
        int carrotPerDay = 5; // В этой переменной фиксируем ежедневный рацион  
  
        // Число итераций совпадает с количеством дней в неделе  
        for (int day = 1; day <= 7; day = day + 1) { // Переменная итерирования - day  
            carrotCount = carrotCount + carrotPerDay; // При каждой итерации плюс 5 морковок  
            // Сколько морковок Байт съедает за разное количество дней  
            System.out.println(day + "-й день, Байт съест " + carrotCount + " морковок.");  
        }  
        // Сколько овощей требуется покупать на неделю  
        System.out.println("Рацион на неделю: " + carrotCount + " морковок.");  
    }  
}
```

```
1-й день, Байт съест 5 морковок.  
2-й день, Байт съест 10 морковок.  
3-й день, Байт съест 15 морковок.  
4-й день, Байт съест 20 морковок.  
5-й день, Байт съест 25 морковок.  
6-й день, Байт съест 30 морковок.  
7-й день, Байт съест 35 морковок.  
Рацион на неделю: 35 морковок.
```

Мы объявили переменную для общего количества моркови **carrotCount** и переменную для ежедневного рациона Байта **carrotPerDay**. Затем мы запустили цикл на семь итераций (неделю), в ходе которого считаем съеденные хомяком овощи — каждый день прибавляется по пять морковок. Всего на неделю нужно 35 морковок. Чтобы такого количества моркови хватило, например, на две недели, ежедневный рацион должен быть другим. Измените в коде начальные значения переменных — число дней и количество морковок — и посмотрите, сколько корнеплодов съедает Байт за разные промежутки времени.

Условия задачи можно усложнить. К примеру, сосед привёз для Байта с дачи немного моркови — теперь у вас есть небольшой запас. Вы хотите посчитать, сколько нужно докупить, чтобы хватило на месяц. Для решения этой задачи добавим в код ещё одну переменную. И вжух:

```
public class Practice {  
    public static void main(String[] args) {  
        int carrotCount = 0; // Переменная для подсчёта общего количества морковок  
        int carrotPerDay = 3; // Ежедневный рацион теперь три морковки  
        int carrotReserve = 23; // Новая переменная! она отражает, что у вас уже есть 23 морковки  
  
        // цикл на месяц  
        for (int day = 1; day <= 31; day = day + 1) {  
            carrotCount = carrotCount + carrotPerDay; // Вычисляем рацион  
        }  
  
        // Вычитаем из общего рациона число морковок, которые есть в запасе  
        System.out.println("Нужно докупить моркови:" + (carrotCount - carrotReserve));  
    }  
}
```

Результат

Нужно докупить моркови:70

В новой переменной **carrotReserve** хранится число морковок, которые есть у вас дома. В цикле считается, сколько их нужно на месяц — это значение сохранится в переменной **carrotCount**. После выполнения цикла из общего числа морковок (**carrotCount**) вычитается их запас (**carrotReserve**). Если вы измените в коде значения переменных **carrotPerDay** и **carrotReserve**, то узнаете, насколько могут быть прожорливы хомяки. Экспериментируйте!

Задача

Если бы дом, который построил Джек, состоял из чисел, то он выглядел бы вот так:

```
Это первый этаж дома, который построил Джек.  
А это 2 этаж, он на один выше, чем этаж 1  
А это 3 этаж, он на один выше, чем этаж 2  
...
```

Постройте с помощью цикла десятиэтажный дом. Последней строкой цикл должен вывести такую:

```
А это 10 этаж, он на один выше, чем этаж 9
```

Задача

```
public class Practice {  
    public static void main(String[] args) {  
  
        System.out.println("Это первый этаж дома, который построил Джек.");  
        // Вместо многоточий добавьте нужные значения  
        for (...; ...; ...) {  
            // Место для вывода  
        }  
    }  
}
```

Решение

```
System.out.println("Это первый этаж дома, который построил Джек.");  
for (int i = 2; i <= 10; i++) {  
    System.out.println("А это " + i + " этаж, он на один выше, чем этаж" + (i-1));  
}
```



Задача

Проектировать лифты — это слишком приземлённо, попробуйте себя в роли космического инженера. Напишите программу для отправки в космос ракеты SpaceY. При ее запуске должен происходить десятикратный обратный отсчёт. Однако учтите, что пилот не любит классический отсчёт из 10 секунд, — он считает своим счастливым числом семь.

Цикл должен начинаться с 70 секунд и идти с шагом 7. Сохраните количество секунд до старта в переменную **secondsBeforeStart** и используйте её как начальное значение переменной итерирования **i**.

Задача

Результат запуска цикла должен получиться таким:

```
До старта SpaceY осталось 70
До старта SpaceY осталось 63
До старта SpaceY осталось 56
До старта SpaceY осталось 49
До старта SpaceY осталось 42
До старта SpaceY осталось 35
До старта SpaceY осталось 28
До старта SpaceY осталось 21
До старта SpaceY осталось 14
До старта SpaceY осталось 7
До старта SpaceY осталось 0
Поехали! Узнаем, есть ли жизнь на Марсе!
```

Задача

```
public class Practice {  
    public static void main(String[] args) {  
        int secondsBeforeStart = 70; // Секунды до старта, цикл должен начинаться с их значения  
  
        for (...; ...; ...) {  
            // Место для вывода  
        }  
  
        System.out.println("Поехали! Узнаем, есть ли жизнь на Марсе!");  
    }  
}
```


Решение

```
public class Practice {  
    public static void main(String[] args) {  
        int secondsBeforeStart = 70;  
  
        for (int i = secondsBeforeStart; i >= 0; i = i - 7) {  
            System.out.println("До старта SpaceY осталось " + i);  
        }  
  
        System.out.println("Поехали! Узнаем, есть ли жизнь на Марсе!");  
    }  
}
```

Задача

Ваш друг планирует купить новые кроссовки и решил ежедневно откладывать на это небольшую сумму. Напишите программу, которая поможет ему следить за накоплениями. С помощью переменной **days** задайте количество дней, в течение которых он будет копить, а сумма ежедневных взносов пусть хранится в переменной **moneyPerDay**. Определите значения этих переменных в коде так, чтобы программа напечатала следующее:

```
День 1. Уже 200 тенге.  
День 2. Уже 400 тенге.  
День 3. Уже 600 тенге.  
День 4. Уже 800 тенге.  
День 5. Уже 1000 тенге.
```

Задача

```
public class Practice {  
  
    public static void main(String[] args) {  
        int days = ...; // Количество дней  
        int moneyPerDay = ...; // Нужно откладывать в день  
        int sum = 0; // Переменная, которая хранит общую сумму накоплений  
  
        for (...; ...; ...) {  
            sum = ...; // Каждый день увеличивается на значение moneyPerDay  
            // место для вывода  
        }  
    }  
}
```

Решение

```
public class Practice {  
  
    public static void main(String[] args) {  
        int days = 5; // Количество дней  
        int moneyPerDay = 200; // Нужно откладывать в день  
        int sum = 0; // Переменная, которая хранит общую сумму накоплений  
  
        for (int i = 1; i <= days; i++) {  
            sum = moneyPerDay * i; // Каждый день увеличивается на значение moneyPerDay  
            // место для вывода  
            System.out.println("День " + i + ". Уже " + sum + " тенге.");  
        }  
    }  
}
```

Задача

Байт тем временем налёг на новый корм — съедает по целой пачке в день. Ветеринар считает, что это не очень полезно, и порекомендовал делить пачку на три дня. Вы, как обычно, отложили на корм 2500 тенге. Подсчитайте с помощью цикла, на сколько дней вам хватит этих денег, если одна пачка корма для Байта стоит 500 тенге. Ваша программа должна выводить каждый трёхдневный период и остаток средств к этому моменту:


На 3-й день останется 2000 тенге.
На 6-й день останется 1500 тенге.
На 9-й день останется 1000 тенге.
На 12-й день останется 500 тенге.
На 15-й день останется 0 тенге.
Денег хватит на 15 дней.

Задача

```
public class Practice {  
    public static void main(String[] args) {  
        int money = 2500; // Деньги на корм  
        int foodPerDay = 500; // Стоимость пачки корма  
        int days = 0; // Дни, когда вы сможете покупать корм  
  
        ... // Здесь опишите цикл, начинаем с 2500 тенге  
        ... // Здесь отразите, что вы будете покупать корм каждые три дня  
        ...// Опишите строку вывода  
        ...// Цикл оканчивается здесь  
  
        System.out.println("Денег хватит на " + ... + " дней.");  
    }  
}
```

Решение

```
public class Practice {  
    public static void main(String[] args) {  
        int money = 2500; // Деньги на корм  
        int foodPerDay = 500; // Стоимость пачки корма  
        int days = 0; // Дни, когда вы сможете покупать корм  
        money -= foodPerDay;  
  
        for (int i = money; i >= 0; i = i - foodPerDay){  
            days += 3;  
            System.out.println("На " + days + "-й день останется " + i + " тенге.");  
        } // Цикл оканчивается здесь  
  
        System.out.println("Денег хватит на " + days + " дней.");  
    }  
}
```



Циклы с условием и вложенные циклы



Циклы с условием и вложенные циклы

Вы уже немало знаете о циклах — умеете с их помощью печатать строки и проводить расчёты. Однако циклы не существуют в программе сами по себе. Их выполнение может зависеть от внешних обстоятельств.

К примеру, благодаря советам ветеринара Байт отлично похудел к лету. Чтобы оставаться в форме, он решил устраивать разгрузочные дни. Каждое утро Байт взвешивается и определяет, будет ли он сегодня есть морковки или только попьёт воды и побегает в колесе. Если вес меньше 800 грамм, то Байт съедает как обычно пять морковок, в обратном случае объявляет разгрузочный день. Чтобы отразить это в коде, нужно добавить к циклу условное выражение:

```
public class Practice {  
    public static void main(String[] args) {  
        int weight = 750; // Объявили переменную веса Байта и присвоили ей значение  
        if (weight < 800) { // Цикл сработает, только если вес Байта меньше 800 грамм  
            for (int i = 1; i < 6; i = i + 1) {  
                System.out.println("Байт съел " + i + "-ю морковку");  
            }  
        } else {  
            System.out.println("Разгрузочный день. Пьём водичку, крутим колесо!");  
        }  
    }  
}
```



Циклы с условием и вложенные циклы

Условные выражения и циклы часто используются вместе. Причём и цикл может находиться внутри условного выражения, как в примере с Байтом, так и условное выражение может быть внутри цикла.

Вопрос

Прочитайте этот код исходя из того, что вы уже знаете о циклах и условных выражениях. Каким будет конечное значение переменной **counter**?

```
public class Practice {  
    public static void main(String[] args) {  
        int counter = 0;  
  
        for (int i = 0; i < 10; i = i + 1) {  
            if (i > 7) {  
                counter = counter + 2;  
            }  
            counter = counter + 1;  
        }  
    }  
}
```

A. 10

B. 13

C. 14

D. 12



Ответ

A. 10

B. 13

C. 14

D. 12

Циклы с условием и вложенные циклы

Ваш ЗОЖ-хомяк решил не останавливаться на достигнутом и просит запрограммировать для него ещё и расписание пробежек в колесе. Байт решил, что следующие три дня хочет тренироваться по два раза — утром и вечером. Нужно учесть эти два параметра: число тренировочных дней и количество пробежек в день, и запрограммировать вот такое расписание:

```
День 1
  Пробежка 1
  Пробежка 2
День 2
  Пробежка 1
  Пробежка 2
День 3
  Пробежка 1
  Пробежка 2
```

Циклы с условием и вложенные циклы

Попробуем справиться с помощью обычного цикла **for** с шагом в единицу:

```
int days = 3; //Число дней с тренировками

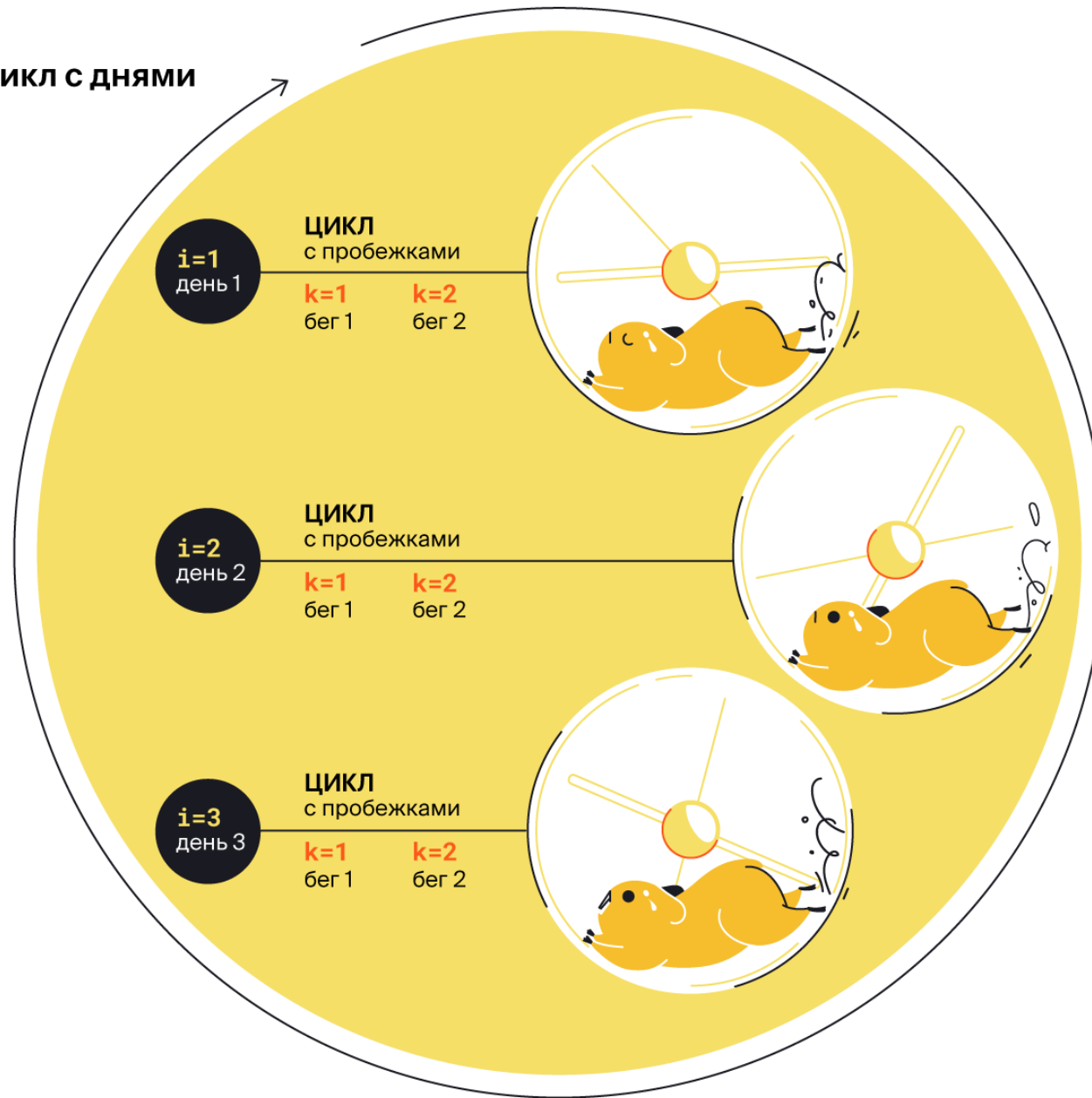
for (int i = 1; i <= days; i++) {
    System.out.println("День " + i); // Смена дней происходит в цикле
    System.out.println("Пробежка 1");
    System.out.println("Пробежка 2");
}
```



Циклы с условием и вложенные циклы

Вроде получилось?! Дни тренировок теперь сменяются в цикле. Однако чтобы отображались пробежки, пришлось дублировать код, а как вы помните — **Don't Repeat Yourself**. Получается, тренировки Байта тоже должны быть в цикле. И так как мы знаем их точное число, для этого также подойдёт цикл **for**.

Цикл с днями



Таким образом, нам нужно, чтобы в цикле для смены дней запускался цикл с пробежками. То есть внутри одного цикла оказывается другой — он называется **вложенным**. В коде это выглядит так:

```
public class Practice {
    public static void main(String[] args) {
        int days = 3; // Количество дней, в которые Байт будет тренироваться
        int run = 2; // Число пробежек в день

        for (int i = 1; i <= days; i++) { // Внешний цикл для смены дней
            System.out.println("День " + i);
            for (int j = 1; j <= run; j++) { // Вложенный цикл для пробежек с новой переменной
                System.out.println("    Пробежка " + j);
            }
        }
    }
}
```



Задача

С понедельника вы твёрдо решили заняться английским. Чтобы прокачать свой уровень, вы планируете в течение 30 дней смотреть в день по три серии любого сериала на английском. Допишите код внешнего и вложенного циклов, исходя из этих вводных:

Задача

```
public class Practice {  
    public static void main(String[] args) {  
  
        for (...; ...; ...) {  
            System.out.println("День " + ...);  
            for (...; ...; ...) {  
                System.out.println(" Серия " + ...);  
            }  
        }  
        System.out.println("Ура! Вы прокачали и Java, и английский!");  
    }  
}
```



Циклы с условием и вложенные циклы

Вложенные циклы — удобный инструмент. Границ воображению здесь нет — можно «вкладывать» один цикл в другой сколько угодно раз. Давайте усложним тренировочную программу Байта, добавив в неё правила питания. Хомяк не щадит себя на тренировках, поэтому после пробежки в колесе ветеринар разрешил ему съесть по пять морковок. Это можно отразить, добавив в код второй вложенный цикл:

```
public class Practice {
    public static void main(String[] args) {

        int days = 3; // Количество дней, когда Байт будет тренироваться
        int run = 2; // Число пробежек в день
        int carrots = 5; // Число морковок после пробежки

        for (int i = 1; i <= days; i++) { // Внешний цикл для смены дней
            System.out.println("День " + i);

            for (int j = 1; j <= run; j++) { // Вложенный цикл для пробежек
                System.out.println(" Пробежка " + j);

                for (int k = 1; k <= carrots; k++) { // Ещё один вложенный цикл для морковок
                    System.out.println(" Морковка " + k);
                }
            }
        }
        System.out.println("Самое время спеть: всё идет по планууу!");
    }
}
```



Циклы с условием и вложенные циклы

При объявлении нескольких вложенных циклов не забудьте, что переменная итерирования каждого из них должна называться по-своему. Если вы назовёте их одинаково, то программа выдаст ошибку. Также важно помнить об областях видимости таких переменных: они видны в блоке своего цикла и вложенных в него, а за пределами этих циклов — нет. То есть в нашем примере переменная **i** видна во всех трёх циклах, а переменная **k** — только внутри цикла с морковками.



Выберите верные определения вложенного цикла.

- A. Это цикл внутри цикла.
- B. Это цикл, в который вложен другой цикл.
- C. Это цикл, в который вложили условие.
- D. Это цикл внутри цикла внутри цикла внутри цикла.
- E. Это дополнительный цикл, который позволяет внутри тела основного цикла повторять какое-то действие.



Ответ

- A. Это цикл внутри цикла.
- B. Это цикл, в который вложен другой цикл.
- C. Это цикл, в который вложили условие.
- D. Это цикл внутри цикла внутри цикла внутри цикла.
- E. Это дополнительный цикл, который позволяет внутри тела основного цикла повторять какое-то действие.



Задача

Сегодня необычный день — курьер привёз посылки для каждого жильца вашего дома! Нельзя пропустить ни одной квартиры — в каждую нужно что-то доставить. Чтобы не запутаться, курьер решил обходить все этажи и квартиры строго последовательно в порядке возрастания.

Всего в доме 10 этажей. На каждом из них по 5 квартир — их число хранится в переменной **flatsNumber**. Используя вложенные циклы, напишите программу для учёта работы курьера, которая будет печатать номера этажей и квартир и сообщение, что посылка доставлена.

Задача

При смене этажа нужно учитывать те квартиры, где курьер уже побывал. Для этого потребуется высчитывать номер текущей квартиры и сохранять его в переменную **currentFlat** — в формуле используйте переменные обоих циклов и **flatsNumber**. В результате для квартир должна получиться последовательность от 1 до 50:

```
Этаж 1 квартира 1 – доставлено.  
Этаж 1 квартира 2 – доставлено.  
Этаж 1 квартира 3 – доставлено.  
Этаж 1 квартира 4 – доставлено.  
Этаж 1 квартира 5 – доставлено.  
Этаж 2 квартира 6 – доставлено.  
Этаж 2 квартира 7 – доставлено.  
...
```

Задача

```
public class Practice {  
    public static void main(String[] args) {  
  
        int flatsNumber = 5; // Количество квартир на этаже  
        int floorsNumber = 10; // Количество этажей  
  
        for (int i...; ...; ...) {  
            for (int j...; ...; ...) {  
                int currentFlat = (... - 1) * ... + ...; // Заготовка формулы для учёта квартир  
                System.out.println("Этаж " + ... + " квартира " + ... + " - доставлено.");  
            }  
        }  
    }  
}
```

Решение

```
public class Practice {
    public static void main(String[] args) {

        int flatsNumber = 5; // Количество квартир на этаже
        int floorsNumber = 10; // Количество этажей

        for (int i = 1; i <= floorsNumber; i++) {
            for (int j = 1; j <= flatsNumber; j++) {
                int currentFlat = (i - 1) * flatsNumber + j;
                System.out.println("Этаж " + i + " квартира " + currentFlat + " - доставлено.");
            }
        }
    }
}
```



Задача

https://github.com/practicetasks/java_tasks/blob/main/loops/task_1/README.md



Решение

<https://gist.github.com/practicetasks/1c8b05a44fa8a1ee9df74c31f8033c99>



Цикл `while`



Цикл `while`

На прошлых уроках мы много говорили о цикле **for** — он подходит для тех случаев, когда заранее известно количество итераций. Но так бывает не всегда. К примеру, вы закупили для Байта корм впрок — неизвестно, на сколько порций его хватит. В этом случае работа цикла будет определяться не числом повторений, а булевым выражением — питомец ест корм до тех пор, пока он в наличии. Если корм кончился — булево выражение стало ложным — цикл завершится. Такой цикл описывается с помощью служебного слова **while** (от англ. *while* — до тех пор, пока).

Цикл `while`

В цикле `for` мы определяли переменную итерирования, условие и шаг цикла. В цикле `while` нам неизвестно число итераций, поэтому не получится задать ни переменную, ни шаг. Мы знаем только условие его работы. Его и указываем после слова `while`, перед телом цикла в фигурных скобках:

```
int foodWeight = 500; // Количество корма

while (foodWeight >= 10) { // Условие работы цикла - пока осталась хотя бы одна порция
    foodWeight = foodWeight - 10; // Байт съедает 10 грамм корма за раз
}

System.out.println("Корм закончился! Пора идти в магазин!");
```



Цикл `while`

Структура циклов `for` и `while` похожа, как и логика их работы — цикл выполняется, пока условие верно. В нашем примере цикл `while` будет работать до тех пор, пока значение переменной `foodWeight` больше или равно 10 — дома есть хотя бы одна порция корма. Как только переменная станет меньше 10 — цикл завершится. После этого будет выполнен следующий код — напечатано предупреждение, что пора идти в магазин.

Цикл `while`

Цикл `while` также можно использовать для расчётов. К примеру, ваш друг хочет накопить 25000 тенге. Он не знает заранее, какую точно сумму сможет откладывать каждый день, но решил, что точно не больше 1500 тенге. Эту задачу можно решить, соединив цикл `while` и генератор случайных чисел — `Random`.

Генератор случайных чисел будет играть в коде роль вашего друга: определять число в диапазоне от 0 до 1500 — именно сколько денег будет отложено за день и сохранено в переменной `moneyToday`. Цикл `while` будет складывать эти деньги в копилку, суммируя их, и получать общую сумму накоплений `moneyTotal`.

Одновременно **while** посчитает количество дней **dayCount** — то есть сколько итераций совершил цикл (сколько случайных чисел сгенерировал **Random**), чтобы получилось 25000. Это выглядит так:

```
import java.util.Random;

public class Practice {
    public static void main(String[] args) {
        Random random = new Random(); // Генерирует случайное число

        int dayCount = 0; // Для учёта дней накоплений
        int moneyTotal = 0; // Суммарное количество накоплений
        int moneyToday; // Сколько откладываем сегодня
        int goal = 25000; // Финансовая цель

        while (moneyTotal <= goal) {
            moneyToday = random.nextInt(1500); // Случайная сумма на сегодня
            moneyTotal = moneyTotal + moneyToday; // Добавили эти деньги в копилку
            dayCount = dayCount + 1; // Засчитали день
        }

        System.out.println("Ура! Вы смогли накопить " + goal + " за " + dayCount + " дней.");
    }
}
```



Цикл `while`

Цикл `while` удобно использовать, когда работа программы зависит от внешних данных, которые изначально неизвестны, — например, их вводит пользователь. Допустим, вы разрабатываете приложение-тренажёр по математике. Предполагается, что пользователь должен решать примеры с экрана и вводить свои ответы. Если ответ неправильный, то требуется повторять пример снова и снова — это и обеспечит терпеливый цикл `while`. В случае правильного ответа цикл завершится.

Цикл `while`

Посмотрим, как это будет выглядеть в коде. Возьмём простой пример «Сколько будет 2+3?» и создадим цикл **`while`**, который будет предлагать пользователю решить его. Нужно, чтобы цикл выполнялся до тех пор, пока не введён правильный ответ, равный 5. Чтобы отразить это в условии цикла, потребуется **логический оператор «не равно» `!=`**. Он работает так: если левая часть не равна правой, то условие истинно. Например, `2 != 5` (два не равно пяти) — истинно, цикл продолжается, а `5 != 5` — ложно (пять равно пяти), цикл завершается.

Цикл while

Кроме того, для считывания ответа понадобится **Scanner** — введённое с консоли число будет храниться в переменной `x`. Таким образом, условие работы цикла получится таким — `x != 5`. Соединяем всё это:

```
Scanner scanner = new Scanner(System.in); // Позволяет считать ответ пользователя
int x = 0; // Хранит ответ пользователя

while (x != 5) { // Цикл выполняется, пока x не равна 5
    System.out.println("Сколько будет 2+3?");
    x = scanner.nextInt(); // Считываем из консоли число и присваиваем переменной x
}
System.out.println("Пример решён правильно!");
```




Цикл `while`

В переменную `x` сохраняется число, которое вводит пользователь. Сначала ей присвоено значение `0` — пользователь ещё не ввёл ничего, однако это нужно, чтобы цикл заработал. В теле цикла даём пример и считываем ответ пользователя с помощью выражения `scanner.nextInt()` — `int` внутри него означает, что нужно считать из консоли именно числовое значение (если ввести буквенное, строку — будет ошибка).



Задача

Накопления — дело серьёзное. Ваш друг уже накопил 5000 тенге и решил замахнуться на миллион. Для этого он положил свои сбережения на накопительный счёт под 5% годовых. Допишите программу с циклом **while**, которая посчитает, через сколько лет на счету будет нужная сумма. Предполагается, что стартовый баланс счёта (сумма сбережений) вводится с консоли.

Задача

```
import java.util.Scanner;

public class Practice {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Сколько денег у вас сейчас: ");
        double balance = scanner.nextDouble(); // Ваши сбережения
        int years = 0;

        while (...) {
            balance = ...;
            years = ...;
        }

        System.out.println("Через " + ... + " лет у вас будет миллион!");
    }
}
```

Решение

```
Scanner scanner = new Scanner(System.in);
System.out.println("Сколько денег у вас сейчас: ");
double balance = scanner.nextDouble(); // Ваши сбережения
int years = 0;

while (balance < 1000000) {
    balance = balance + balance * 0.05;
    years += 1;
}

System.out.println("Через " + years + " лет у вас будет миллион!");
```




Задача

Пришло время сыграть в игру! Но сначала вам придётся её запрограммировать. Правила следующие: вы против компьютера, компьютер «загадывает» число от 0 до 1000, а вам нужно угадать это число. Было бы негуманно делать это без подсказок. На каждый ваш вариант компьютер должен говорить либо «Больше», если загаданное число больше, или «Меньше», если наоборот.



Задача

В прекоде уже отражено, что компьютер будет загадывать числа с помощью генератора **Random** и сохранять их в переменную **randomInt**. Требуется дописать цикл **while** — игра должна длиться до тех пор, пока вы не отгадаете загаданное компьютером число. Ваши варианты сохраняются в переменной **userInput** — её нужно считывать из консоли внутри цикла. Подсказки оформите в коде с помощью условных выражений.



Например, если компьютер загадал число 407, один из вариантов работы программы получится таким:

Я загадал число от 0 до 1000.

Ваш ход:

500

Меньше

250

Больше

400

Больше

450

Меньше

410

Меньше

405

Больше

407

Вы великолепны! Именно это я загадал.

```
public class Practice {  
    public static void main(String[] args) {  
  
        Scanner scanner = new Scanner(System.in);  
        int randomInt = new Random().nextInt(1000); // Генерирует новое число от 0 до 1000  
  
        int userInput = -1; // Это нужно, чтобы цикл запустился, если Random выдаст 0  
        System.out.println("Я загадал число от 0 до 1000.");  
        System.out.println("Ваш ход:");  
  
        // Напишите условие цикла для запуска игры  
  
        userInput = ...; // В этой переменной должен сохраняться ввод пользователя  
  
        if (...) { // Условие проверяется в цикле  
            System.out.println("Меньше");  
        }  
        // Второе условие  
        System.out.println("Больше");  
  
        // Печатаем, когда число угадано  
        System.out.println("Вы великолепны! Именно это я загадал.");  
    }  
}
```


Решение

```
while (randomInt != userInput){  
  
    userInput = scanner.nextInt(); // В этой переменной должен сохраняться ввод  
пользователя  
  
    if (userInput > randomInt) { // Условие проверяется в цикле  
        System.out.println("Меньше");  
    } else if (userInput < randomInt){  
        System.out.println("Больше");  
    }  
}  
  
System.out.println("Вы великолепны! Именно это я загадал.");
```



Бесконечный цикл

Бесконечный цикл

Продолжаем изучать возможности цикла **while**. Он выполняется до тех пор, пока условие является истинным. Следовательно, если в условии написать **true**, то цикл будет выполняться снова и снова. Посмотрим, что получится, если запустить такой цикл для пробежки Байта в колесе.

```
while (true) {  
    System.out.println("Бегу, бегу, бегу!"); // Будет выполняться бесконечно  
}  
System.out.println("Отлично побегал!"); // Эта строка не будет напечатана
```



Бесконечный цикл

В консоли будет непрерывно печататься строка "Бегу, бегу, бегу!", а до строки "Отлично побегал!" очередь никогда не дойдёт. Хомяк Байт в колесе превратился в вечный двигатель! Такой цикл, который начинается со служебного слова **while** и содержит в условии **true**, называется **бесконечным**. И никакого символа бесконечности ∞ не требуется — его просто-напросто нет в синтаксисе Java.

Бесконечный цикл

Если же написать в условии цикла **false** и попробовать запустить код, то вообще ничего не получится. Программа выдаст ошибку: **unreachable statement** (англ. «недостижимое утверждение»).

```
while (false) {  
    System.out.println("Никуда не побегу!"); // Такой цикл не запустится. Хомяк решил  
    поспать.  
}
```



Бесконечный цикл

Бесконечные циклы активно используются, например, при передаче данных, когда их объём неизвестен заранее. Или в построении web-приложений, которые общаются через не всегда надёжную сеть и могут «падать», и поэтому им требуется снова и снова отправлять друг другу запросы. При работе с пользователями бесконечные циклы нужны, чтобы обеспечить непрерывное ожидание и обработку таких операций, как касание экрана, свайпы или ввод текста и других.

Выберите правильные варианты того, как создать бесконечный цикл?

- A. Использовать `for (int i = 0; i \neq ∞ ; i++)`
- B. Использовать `while (∞)`
- C. Использовать `for (int i = 0; i < 100000000000; i++)`
- D. Использовать `while (true)`
- E. Использовать `while (false)`

Выберите правильные варианты того, как создать бесконечный цикл?

- A. Использовать `for (int i = 0; i \neq ∞ ; i++)`
- B. Использовать `while (∞)`
- C. Использовать `for (int i = 0; i < 100000000000; i++)`
- D. Использовать `while (true)`
- E. Использовать `while (false)`



Бесконечный цикл

Бесконечный цикл с прерыванием **break** нужен, когда требуется выполнять условно неограниченное количество итераций до достижения определённого результата.

Например, при вызове такси в приложении радиус поиска свободной машины расширяется до тех пор, пока не найдётся первый подходящий автомобиль. Однако условий выхода из цикла (подходящих машин) может быть несколько, и в таком случае мы можем предложить пользователю выбор.



Бесконечный цикл

Пользователю предлагается оценить работу приложения для вызова такси по шкале от одного до трёх. Для каждой оценки предусмотрен свой вариант ответа. После того как пользователь ввёл 1, 2 или 3, программа будет печатать соответствующий ответ и завершать цикл с помощью `break`. Если введены другие цифры — вопрос будет повторяться в бесконечном цикле. Позапускайте приложение и поставьте разные оценки, в том числе за пределами требуемого диапазона. Программа будет снова и снова предлагать вам выбрать только 1, 2 или 3 — это всё заслуга бесконечного цикла. В этом тренажёре нет правильных ответов — мы добавили его специально, чтобы вы смогли попробовать, как работает бесконечный цикл с прерыванием `break`.

```
public static void main(String[] args) {

    Scanner scanner = new Scanner(System.in);
    System.out.println("Оцените работу приложения Зеленоглазое-Такси по шкале от 1 до 3");

    while (true) { // Запускаем бесконечный цикл
        int x = scanner.nextInt(); // Получаем оценку от пользователя

        if (x == 1) { //Если пользователь вводит 1
            System.out.println("Спасибо за вашу оценку. Мы станем лучше!"); // Благодарим
            break; // Выходим из цикла
        }
        if (x == 2) { // Если оценка равна 2
            System.out.println("Спасибо за вашу оценку. Есть куда расти!");
            break; // И выходим из цикла
        }
        if (x == 3) { // Если оценка равна 3
            System.out.println("Спасибо за высокую оценку! Рады, что вы выбрали нас!");
            break; // И выходим из цикла
        }
        // Пользователь ввёл не 1, 2 или 3 - повторяем просьбу
        System.out.println("Ваша оценка должна быть в диапазоне от 1 до 3");
    }
}
```



Задача

Бесконечный цикл `while (true)` с прерыванием `break` применяется для решения достаточно ограниченного спектра задач. Чаще всего его удобнее заменить на цикл `while` с условием. Потренируйтесь это делать. Эта программа с помощью бесконечного цикла суммирует числа, которые вводит пользователь. Она работает до тех пор, пока не будет введён 0. Перепишите её, заменив бесконечный цикл на цикл `while` с условием.

```
public class SumNumbers {  
    public static void main(String[] args) {  
        Scanner scanner = new Scanner(System.in);  
        int sum = 0;  
        int inputNumber;  
  
        System.out.println("Введите числа для суммирования (0 для остановки):");  
  
        // Получение первого числа перед циклом  
        inputNumber = scanner.nextInt();  
  
        // Цикл продолжается до ввода 0  
        while (inputNumber != 0) {  
            sum += inputNumber;  
            inputNumber = scanner.nextInt(); // Получение следующего числа внутри цикла  
        }  
  
        System.out.println("Сумма введенных чисел: " + sum);  
    }  
}
```

Задача

В одном из первых уроков про циклы вы запустили ракету SpaceY в космос. Сейчас она бесконечно летает по орбите. Чтобы прервать её полет и вернуть на Землю, пилоту нужно ввести секретный код — случайное число от 0 до 100. Проблема в том, что этот код автоматически обновляется в штабе на Земле каждый раз, когда ракета сделала полный круг, а связь потеряна. Пилоту ничего не остаётся, кроме как пытаться отгадать код. Допишите программу таким образом, чтобы, когда пилот отгадает код, полёт ракеты прервался, и он смог полететь домой. Результат работы программы должен быть таким:

```
Ракета SpaceY на орбите!  
Ракета SpaceY на орбите!  
Ракета SpaceY на орбите!  
Ракета SpaceY на орбите!  
...  
Пилот угадал число! Летим домой!
```

```
import java.util.Random;

public class Practice {
    public static void main(String[] args) {
        Random random = new Random(); // Генерирует случайное число
        int secretCode;
        int pilotInput;

        // Добавьте цикл - ракета летает бесконечно
        secretCode = random.nextInt(100); // Здесь задаётся код - случайное число от 0 до 100
        System.out.println("Ракета SpaceY на орбите!");
        pilotInput = random.nextInt(100); // Пилот пытается угадать код


        if (...) { // Если пилот угадал код, то цикл должен завершиться
            System.out.println("Пилот угадал число! Летим домой!");
            ...
        }
    }
}
```

Решение

```
Random random = new Random(); // Генерирует случайное число
int secretCode;
int pilotInput;

while (true){
    // Добавьте цикл - ракета летает бесконечно
    secretCode = random.nextInt(100); // Здесь задаётся код - случайное число от 0 до 100
    System.out.println("Ракета SpaceY на орбите!");
    pilotInput = random.nextInt(100); // Пилот пытается угадать код

    if (secretCode == pilotInput) { // Если пилот угадал код, то цикл должен завершиться
        System.out.println("Пилот угадал число! Летим домой!");
        break;
    }
}
```

Прокачиваем финансовое
приложение и добавляем в него
циклы

Прокачиваем финансовое приложение и добавляем в него циклы

Вам нужно улучшить работу финансового приложения, чтобы оно стало удобным для пользователя.

- Создайте цифровое меню для выбора валют при конвертации, где 1 — USD, 2 — EUR, 3 — JPY.
- Добавьте в код приложения бесконечный цикл **while** и предусмотрите выход из него при выборе пользователем команды 0.

https://github.com/practicetasks/java_tasks/tree/main/loops/task_2



Решение

<https://gist.github.com/practicetasks/c649caffc730a499d7f5a81b398de842>