



План занятия

1. [Условные выражения](#)
2. [Больше ветвлений: if - else if](#)
3. [Вложенные условия](#)
4. [Импорт пакетов и монтаж приложения](#)



Условные выражения

Условные операторы

Чаще всего логический тип данных (**boolean**) применяют в **условных выражениях**. Например, вы приходите на рынок и хотите купить яблок. У вас есть одно условие — они должны быть красными. С яблоками или без, но после рынка вы точно пойдёте домой.



Условные операторы

В коде описать подобную последовательность действий можно с помощью условных выражений:

```
if (условие) {  
    // Код #1 выполнится, только если условие == true  
}  
// Код #2 выполнится в любом случае
```

Условные операторы

Оператор `if` (англ. «если») можно прочесть так: «**ЕСЛИ** условие истинно, **ТО** выполнить код #1». В круглых скобках указывается выражение с типом **`boolean`**, а в фигурных пишутся команды, которые должны быть исполнены, если условие истинно. Вы уже сталкивались с фигурными скобками: они нужны, чтобы группировать команды. Такая последовательность команд называется **блоком кода**. Как только код внутри блока выполнится, программа перейдёт к коду #2, идущему после условного выражения. Так, в примере после покупки красных яблок вы пойдёте домой.

Если же проверяемое условие ложно, программа сразу начнёт выполнять код #2, то есть никаких яблок вы не купите, а сразу вернётесь домой с пустыми руками.

Условные операторы

```
public class Practice {  
    public static void main(String[] args) {  
        String color = "Зелёный";  
        if (color.equals("Красный")) {  
            System.out.println("Беру яблоки!");  
        }  
        System.out.println("Иду домой.");  
    }  
}
```

Условные операторы

У блока кода есть важная особенность. Если вы объявите переменную внутри него, то она не будет видна за его пределами, снаружи этого блока. Это называется **областью видимости переменных**. Рассмотрим пример:

```
boolean condition = true;

if (condition) {
    double rateUSD = 444.06;
    System.out.println("Внутри блока курс доллара виден. Он равен " + rateUSD);
}

// Здесь переменная rateUSD не видна
System.out.println("Вне блока курс доллара не виден: " + rateUSD); // Произойдёт ошибка
```



Условные операторы

Если попытаться запустить этот код, то ничего напечатано не будет. Произойдёт ошибка, потому что переменная `rateUSD` вне блока `if` не видна.

Обратите внимание и на особенность оформления — отступы: весь код внутри блока должен быть сдвинут на 4 пробела вправо. Это не строгое правило, но так гораздо проще читать код.

Условные операторы

Выберите имена переменных, которые не будут видны за пределами блока **if**.

```
int variable1 = 0;
double variable2;
int variable3 = 3;

if (variable3 < 5) {
    String variable4 = "С областями видимости у новичков часто бывают сложности.";
    double variable5 = 76.5;
    variable1 = 33;
    System.out.println("Но у вас таких проблем не будет!");
}

double variable6;
```



Условные операторы

Выберите имена переменных, которые не будут видны за пределами блока **if**.

A) `variable1`

B) `variable2`

C) `variable3`

D) `variable4`

E) `variable5`

F) `variable6`



Условные операторы

Выберите имена переменных, которые не будут видны за пределами блока `if`.

D) `variable4`

Правильно! Эту переменную вы можете использовать только внутри блока `if`.

E) `variable5`

И эта тоже не будет видна за пределами блока кода.

Условные выражения с if-else

Допустим, вы решили, что купите яблоки, **если** они красные, а **иначе** вы купите апельсины.

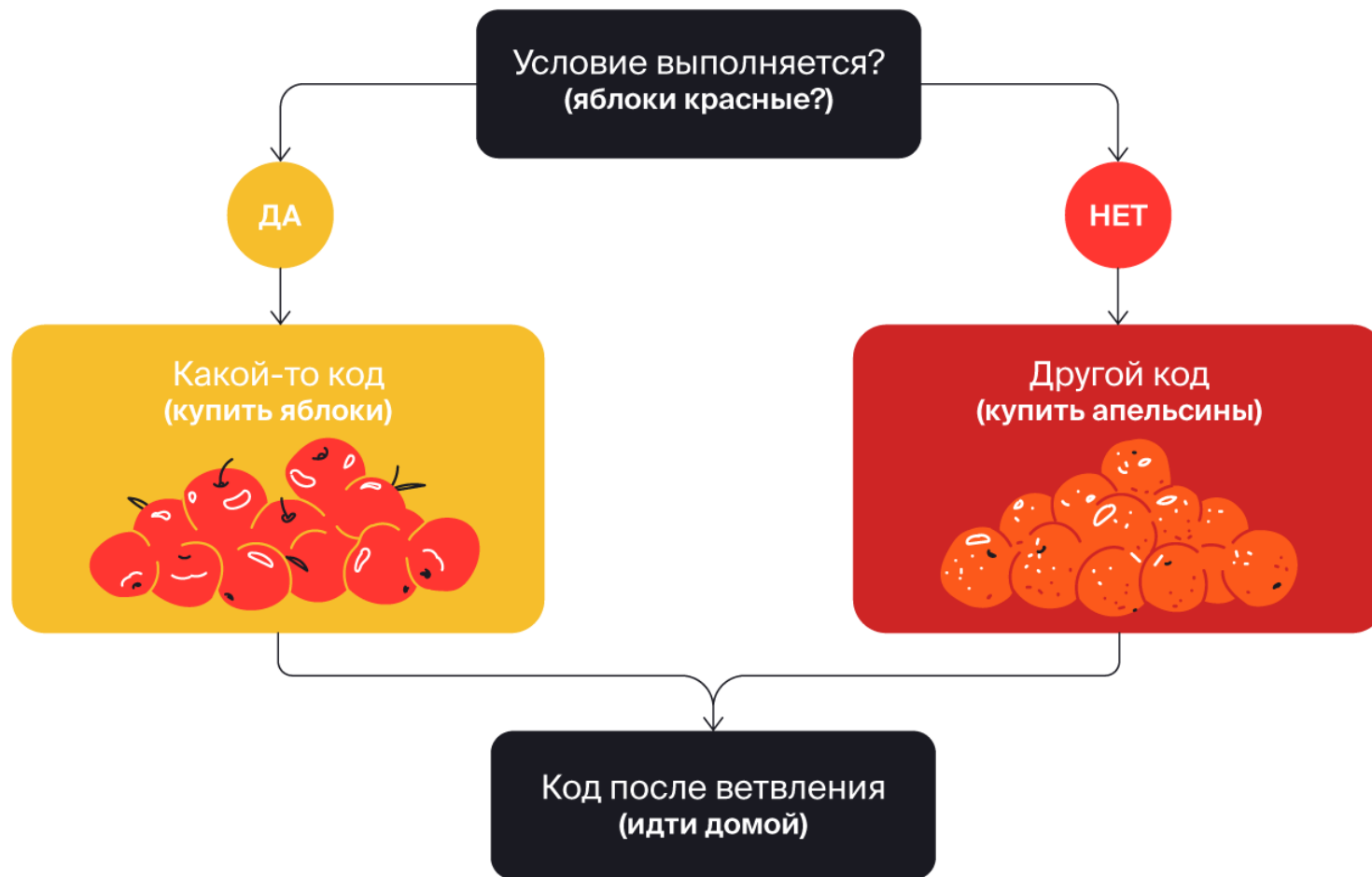


Условные выражения с if-else

В коде это можно описать с помощью оператора **else** (англ. «иначе»). Код внутри этого блока будет выполняться, если условие в скобках не истинно (возвращает **false**):

```
if (условие) {  
    // Код #1 выполнится, только если условие == true  
} else {  
    // Код #2 выполнится, только если условие == false  
}  
  
// Код #3 выполнится в любом случае
```

Такой код можно прочесть следующим образом: «**ЕСЛИ** условие истинно, **ТО** выполнить код внутри блока **if** (купить яблоки), **ИНАЧЕ** выполнить код внутри блока **else** (купить апельсины). Потом продолжить выполнение кода, находящегося после условного выражения (идти домой)».



Условные выражения с if-else

Сделаем нашего советчика более заботливым с помощью условного оператора. Теперь приложение сможет давать вам дельные финансовые советы:

```
public class Practice {  
    public static void main(String[] args) {  
        double rateUSD = 444.06;  
  
        if (rateUSD <= 150) {  
            System.out.println("Кажется, вы изобрели машину времени... Скупайте доллары!");  
        } else {  
            System.out.println("Ничего страшного! Купите немного валюты сейчас.");  
        }  
  
        System.out.println("Советуем регулярно покупать валюту, чтобы не грустить потом. ;)");  
    }  
}
```

Задача 1

Исправьте программу так, чтобы рекомендация по покупке долларов печаталась, только если курс меньше 450. Не забудьте проверить, что всё работает — поменяйте значение курса и убедитесь, что программа может дать вам совет.

```
public class Practice {  
    public static void main(String[] args) {  
  
        double rateUSD = 444.06;  
        double tenges = 124356.5;  
  
        ...  
        System.out.println("Отличный курс доллара – рекомендую купить!");  
        ...  
  
        System.out.println("Ваши сбережения в долларах: " + tenges / rateUSD);  
    }  
}
```


Решение

```
public class Practice {  
    public static void main(String[] args) {  
  
        double rateUSD = 444.06;  
        double tenges = 124356.5;  
  
        if (rateUSD < 450) {  
            System.out.println("Отличный курс доллара – рекомендую купить!");  
        }  
  
        System.out.println("Ваши сбережения в долларах: " + tenges / rateUSD);  
    }  
}
```

Задача 2

Давайте ещё немного прокачаем приложение. Сделаем так, чтобы оно могло конвертировать не только в доллары, но и в евро. В переменной **currency** сохраните название валюты: **USD** или **EUR**. В зависимости от того, какая валюта хранится в переменной, пользователь увидит сумму своих сбережений в долларах или евро.

```
public class Practice {  
    public static void main(String[] args) {  
  
        double rateUSD = 444.06;  
        double rateEUR = 489.32;  
        double tenges = 124356.5;  
  
        String currency = "USD";  
        System.out.println("Вы конвертируете тенге в " + currency);  
        // Если currency равно "USD", то конвертируем в доллары, иначе в евро  
        ... {  
            System.out.println("Ваши сбережения в долларах: " + tenges / rateUSD);  
        } ... {  
            System.out.println("Ваши сбережения в евро: " + tenges / rateEUR);  
        }  
    }  
}
```

Решение

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

        double rateUSD = 444.06;
        double rateEUR = 489.32;
        double tenges = 124356.5;

        String currency = "USD";
        System.out.println("Вы конвертируете тенге в " + currency);
        // Если currency равно "USD", то конвертируем в доллары, иначе в евро
        if (currency.equals("USD")) {
            System.out.println("Ваши сбережения в долларах: " + tenges / rateUSD);
        } else {
            System.out.println("Ваши сбережения в евро: " + tenges / rateEUR);
        }
    }
}
```



Инкремент и декремент

Инкремент и декремент

Как увеличить переменную `i` на единицу?

```
int i = 0;  
i = i + 1;
```

Инкремент и декремент

Теперь вы знаете, что эти операции можно записать ещё и вот так:

```
i += 1;  
i -= 1;
```

Это сокращенная версия прошлого примера на слайде 21.

Однако с помощью операторов `++` и `--` эти выражения можно записать ещё короче:

```
i++;  
i--;
```



Инкремент и декремент

У выражений с `++` и `--` есть и другие важные особенности.


Операция, увеличивающая переменную на единицу, называется **инкрементом** (от англ. *increase* — «увеличивать»), а операция, уменьшающая переменную на единицу — **декрементом** (англ. *decrease* — «уменьшать»). Соответственно, `++` и `--` называются **операторами инкремента и декремента**.



Инкремент и декремент

Их можно записать как после переменной, так и перед ней:

```
++i;  
--i;
```

Запись операторов инкремента и декремента после переменной называют «**постфиксной**», а до — «**префиксной**». Выбор записи определяет порядок выполнения действий в коде. Префиксные операции всегда выполняются первыми в строке:

```
public class Practice {  
    public static void main(String[] args) {  
        int prefix = 20;  
        System.out.println(prefix);    // здесь значение равно 20  
        System.out.println(++prefix); // значение сначала станет 21, а потом будет напечатано  
    }  
}
```



Результат

```
20  
21
```

В последней строке сначала сработает префиксная операция инкремента — изменится значение переменной **prefix**, а только потом печать. В результате в консоли появится значение 21.

Постфиксные операции инкремента и декремента, наоборот, выполняются после других действий в строке:

```
public class Practice {  
    public static void main(String[] args) {  
        int postfix = 10;  
        System.out.println(postfix);    // значение равно 10  
        System.out.println(postfix++); // сначала будет напечатано 10, потом прибавится единица  
        System.out.println(postfix);    // значение postfix теперь равно 11  
    }  
}
```



Результат

```
10  
10  
11
```

Постфиксный инкремент сработает уже после того, как значение **postfix** напечатано — поэтому в консоли во второй строке будет 10, а не 11. Во всех следующих командах будет использоваться уже новое значение. Измените запись инкремента на префиксную и посмотрите, что напечатает программа.



Остаток от деления



Остаток от деления

Ещё одна распространённая арифметическая операция в программировании — получение остатка от деления. Представьте, курьер привёз пиццу из семи кусочков для компании из трёх человек. Каждый съел по два кусочка, в коробке остался один ничейный — это и есть остаток от деления.

В коде получение остатка от деления обозначается операторами % и %=:

```
public class Practice {  
    public static void main(String[] args) {  
        int a;  
        int b = 120;  
        int c = 50;  
        a = b % c; // остаток от деления  
        System.out.println(a);  
        int d = 3;  
        a %= d; // сокращённая запись для a = a % d  
        System.out.println(a);  
    }  
}
```




Результат

20
2

Сначала **a** будет равна 20 — то есть в 120 вмещается два раза по 50 и ещё остаётся 20. Остаток от деления **a** на **d** будет равен 2 — то есть в 20 содержится шесть раз по 3 и ещё остаётся 2. В виде формулы это выглядит так:

$$120 \% 50 = 20$$

120		120 / 50 = 2		50		20
Делимое	—	Частное, полученное при делении нацело	*	Делитель	=	Остаток от деления



Остаток от деления может быть равен нулю. Например, так как 100 делится на 10 без остатка, в `a` сохранится 0:

```
public class Practice {  
    public static void main(String[] args) {  
        int a = 100;  
        a %= 10;  
        System.out.println(a);  
    }  
}
```




Результат

0

Таким образом, с помощью проверки остатка от деления можно понять, является ли одно число делителем другого. Это может пригодиться в решении некоторых задач. Например, если остаток от деления на 2 равен нулю, то сразу понятно, что число чётное.

Допишите код метода, который проверяет, чётное число или нет.

```
public class Practice {  
    public static void main(String[] args) {  
        int number = 10;  
  
        if (number % 2 == 0) {  
            System.out.println(number + " - чётное");  
        } else {  
            System.out.println(number + " - нечётное");  
        }  
    }  
}
```



Остаток от деления можно вычислить и в тех случаях, когда делитель больше делимого. В этом случае его значение будет равно делимому:

```
public class Practice {  
    public static void main(String[] args) {  
        int x = 14;  
        int y = 20;  
        int z = x % y;  
        System.out.println(z);  
    }  
}
```



Результат

14

Логика вычислений такая: в 14 никак не поместится 20, то есть при делении нацело будет ноль. Поэтому остаток от деления получится равным 14.

Вопрос

Чему равно значение переменной **d**?

```
int a = 25 % 5;  
int b = 16 % 3;  
int c = 17 % 20;  
int d = a + b + c;
```



Ответ

18



Порядок вычислений



Порядок вычислений

Чтобы правильно составить арифметическое выражение из нескольких операций, нужно понимать, в каком порядке они будут выполняться.



Порядок вычислений

Порядок выполнения арифметических операций в программировании основывается на математических правилах:

1. Сначала выполняются умножение и деление;
2. Затем — сложение и вычитание;
3. При наличии скобок действия в них выполняются первыми;
4. Считаём всегда слева направо.

Попробуйте посчитать значение переменной **result** сначала самостоятельно и после этого запустите код, чтобы узнать ответ:

```
public class Practice {  
    public static void main(String[] args) {  
        int a = 15;  
        int b = 10;  
  
        int result = a + b * 2 / (2 + 2) - 1;  
        System.out.println(result);  
    }  
}
```



Результат

19

Сначала выполняется сложение в скобках — $(2 + 2) = 4$. Затем слева направо умножение $b * 2 = 10 * 2 = 20$ и деление — 20 нужно разделить на 4 — $20 / 4 = 5$. После этого слева направо сложение $a + 5 = 15 + 5 = 20$ и вычитание единицы — $20 - 1 = 19$.
Результат равен 19.

Чем больше уроков и учебных задач, тем больше выпитых чашек кофе и переменных в коде. На этой неделе ваш одноклассник Игорь прошёл 8 уроков, решил 6 задач и выпил 3 чашки кофе. Определите, сколько переменных он объявил в коде — решите пример, используя математический порядок действий.

```
int classes = 8;  
int tasks = 6;  
int coffeeCups = 3;  
int variables = coffeeCups - classes * tasks + tasks / coffeeCups + (classes +  
coffeeCups) * tasks;
```

- A. `int variables = -192;`
- B. `int variables = 0;`
- C. `int variables = 23;`
- D. `int variables = 18;`
- E. `int variables = -17;`



Ответ

- A. `int variables = -192;`
- B. `int variables = 0;`
- C. `int variables = 23;`
- D. `int variables = 18;`
- E. `int variables = -17;`




Порядок вычислений

При составлении арифметического выражения в коде нельзя ограничиться только математическими правилами. Нужно также учитывать порядок выполнения операторов инкремента и декремента.

Порядок вычислений

Поэтому правила выполнения вычислений в Java можно сформулировать так:

1. Первыми отработают операторы префиксного инкремента и декремента — **++a** и **--a**;
2. Затем — операторы умножения, деления и остатка от деления: *****, **/** и **%**;
3. Третьи на очереди — операторы сложения и вычитания **+** и **-**;
4. Последними выполняются операции с постфиксным инкрементом и декрементом: **a++** и **a--**.



Порядок расположения в коде операторов не отменяет математических правил о том, что вычисления выполняются слева направо, а операции в скобках имеют больший приоритет в сравнении с остальными и выполняются первыми. Рассмотрим на примере:

```
int a = 12;  
int b = 4;  
int d = --a % b + 5 * 4 + b++;
```

Результат

27

В Сначала выполнится префиксный декремент ($--a$) — теперь $a = 11$. Следующим шагом будет взятие остатка: $11 \% 4 = 3$. Дальше идёт умножение и только после этого — сложение: $3 + 20 + 4 = 27$. В результате в переменную a будет сохранено значение 27 и только после этого выполнится инкремент b . Получается, он никак не влияет на результат выражения.

Выберите порядковый номер у каждой операции в выражении для вычисления **awesomeVariable**.

```
int coolVariable = -1;  
int perfectVariable = 7;  
int awesomeVariable = (--coolVariable * 10) + (--perfectVariable + coolVariable);
```

префиксный декремент **--coolVariable** - ?

сложение результатов в скобках - ?

префиксный декремент **--perfectVariable** - ?

сложение **coolVariable** и **perfectVariable** - ?

умножение **coolVariable** на **10** - ?



Ответ


префиксный декремент `--coolVariable` - 1

сложение результатов в скобках - 5

префиксный декремент `--perfectVariable` - 3

сложение `coolVariable` и `perfectVariable` - 4

умножение `coolVariable` на 10 - 2



Если в арифметическом выражении использованы операции инкремента и декремента, то важно учитывать, что они всегда меняют исходное значение переменной. Эти операции сами по себе являются законченными выражениями и для программы нет разницы, стоят они отдельно или внутри каких-либо вычислений:


```
public class Practice {  
    public static void main(String[] args) {  
        int a = 5;  
        --a; // программа вычитет единицу из a и сохранит в неё новое значение  
        int b = --a * 10 + 24; // здесь у a также изменится значение на единицу  
        System.out.println(a);  
    }  
}
```



Результат

3

В результате в переменную **a** будет сохранено значение 3.



Операции с постфиксным инкрементом и декрементом выполняются только после того, как все другие завершены. Поэтому они не влияют на общий результат выражения, а меняют только значение своей переменной. Например, разберём, какие значения будут у каждой из переменных в результате выполнения такого кода:

```
public class Practice {  
    public static void main(String[] args) {  
        int a = 10;  
        int b = 22;  
        int c = (--b / 2) - a++ * 2;  
        System.out.println("a = " + a);  
        System.out.println("b = " + b);  
        System.out.println("c = " + c);  
    }  
}
```


Результат

```
a = 11  
b = 21  
c = -10
```


Вычисления начинаются внутри скобок: Java отнимает единицу у исходного значения `b`, получает 21 и сохраняет его обратно в `b`. Так значение `b` становится равно 21 и больше не меняется.

21 делится на 2 и округляется до целого — остаётся 10. Затем `a` умножается на 2, получается 20. От 10 отнимается 20. В переменную `c` сохраняется значение -10.

Постфиксный инкремент $a++$ не влияет на значение c , так как выполняется после того, как её вычисления завершены. Меняется только исходное значение a . В итоге оно равно 11.

	Значение a	Значение b
$c = (\overbrace{--b}^{21} / 2) - a++ * 2 ;$	10	22
$c = (\overbrace{21}^{10} / 2) - a++ * 2 ;$	10	21
$c = 10 - \overbrace{a++ * 2}^{20} ;$	10	21
$c = \overbrace{10 - 20}^{-10} ; \quad a++ ;$	10	21
$c = -10 ; \quad \overbrace{a++}^{11} ;$	11	21

* Команда выполнится отдельно, в самом конце →



Логические выражения. Утверждение и отрицание

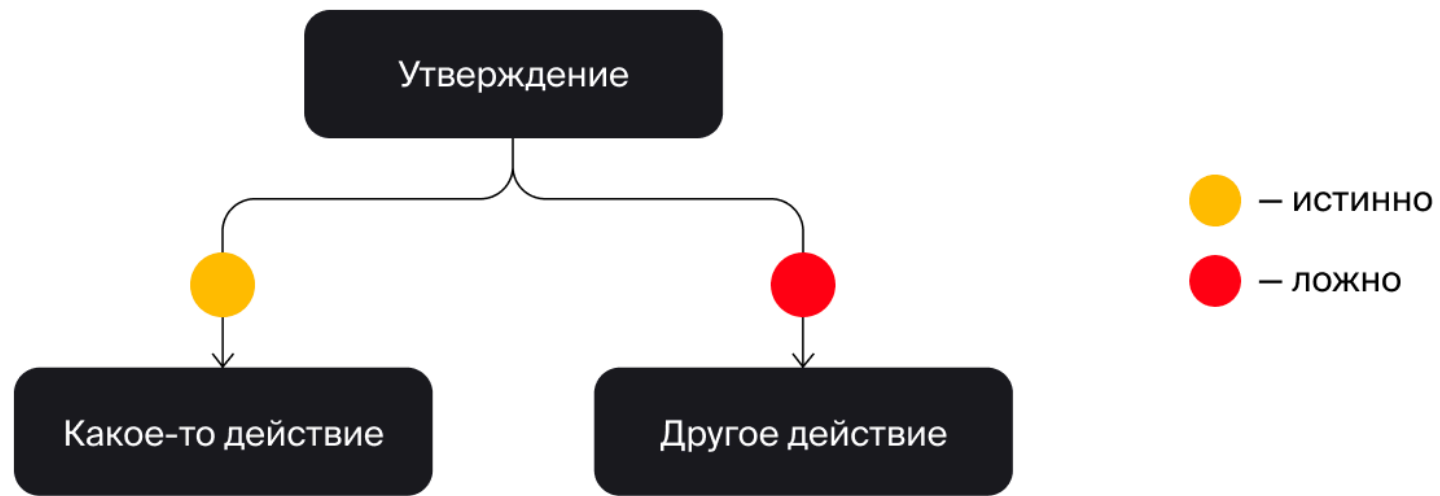



Логические выражения. Утверждение и отрицание

С логическими переменными типа `boolean` также можно составлять выражения. Это часто необходимо при написании ветвлений — булевы выражения используются в условиях `if-else`.

Утверждение

В переменную типа **boolean** чаще всего записывается некое утверждение. Оно может быть либо истинным — тогда переменная принимает значение **true**, либо ложным — значение переменной будет **false**. На основе утверждения можно составить логическую конструкцию — ветвление.





Например, если на улице идёт дождь, то вряд ли стоит идти гулять. В коде это можно отразить так:


```
if (weather.equals("Дождь")) { // если утверждение истинно
    // ничего не делаем, сидим дома
} else { // если утверждение ложно
    System.out.println("Идём гулять!");
}
```

Логическое выражение `weather.equals("Дождь")` — пример утверждения типа **boolean**.

Отрицание


Чтобы изменить значение логического выражения на противоположное, нужно использовать **отрицание**. Оно обозначается **оператором !** перед утверждением. Отрицание можно применить к любому логическому выражению. Например, при создании новой переменной:

```
boolean isPositive = x > 0;  
boolean zeroOrNegative = !(isPositive);
```




Отрицание в условии ветвления предполагает, что требуется что-то сделать только в том случае, если утверждение ложно. Вернёмся к нашему примеру: если идёт дождь (утверждение) — не нужно ничего делать, а вот если нет дождя (отрицание) — идём гулять (действие). В таком случае можно упростить код через отрицание:

```
if (!weather.equals("Дождь")) { // проверяем, что погода - НЕ дождь
    System.out.println("Идём гулять!");
} // блок else больше не нужен
```

Инвертировать можно в том числе и арифметическую операцию. К примеру, возьмём ветвление, где в условии сравниваются числа:

```
if (temperature > 10) {  
} else {  
    System.out.println("Окей, пора надевать пальто!");  
}
```



Надевать пальто следует, только если температура ниже 10 градусов — это можно записать в коде через отрицание. Чтобы его применить, нужно заключить всё выражение в скобки, а оператор ! поставить перед ними:

```
if (!(temperature > 10)) {  
} else {  
    System.out.println("Окей, пора надевать пальто!");  
}
```



Но не всегда стоит увлекаться отрицанием. Иногда достаточно поменять знак в выражении:

```
if (temperature <= 10) {  
    System.out.println("Окей, пора надевать пальто!");  
}
```

Логические выражения `temperature <= 10` и `!(temperature > 10)` равнозначны между собой, но первое воспринимать проще. Аналогично с отрицанием равенства: `!(code == 9999)` можно заменить на более удобную запись с оператором неравенства `code != 9999`.



Какой из вариантов противоположен этому логическому выражению?

```
if (weight < 100) {  
    System.out.println("Надо срочно покормить хомяка!");  
} else {  
    System.out.println("Вес хомяка в норме, кормить не требуется.");  
}
```

A.

```
if (weight >= 100) {  
    System.out.println("Вес хомяка в норме, кормить не требуется.");  
} else {  
    System.out.println("Надо срочно покормить хомяка!");  
}
```

B.

```
if (NOT(weight < 100)) {  
    System.out.println("Вес хомяка в норме, кормить не требуется.");  
} else {  
    System.out.println("Надо срочно покормить хомяка!");  
}
```

C.

```
if (NOT(weight < 100)) {  
    System.out.println("Вес хомяка в норме, кормить не требуется.");  
} else {  
    System.out.println("Надо срочно покормить хомяка!");  
}
```

D.

```
if (!(weight < 100)) {  
    System.out.println("Вес хомяка в норме, кормить не требуется.");  
} else {  
    System.out.println("Надо срочно покормить хомяка!");  
}
```

Ответ

D.

```
if (!(weight < 100)) {  
    System.out.println("Вес хомяка в норме, кормить не требуется.");  
} else {  
    System.out.println("Надо срочно покормить хомяка!");  
}
```



Логические И и ИЛИ



Логические И и ИЛИ


В повседневной жизни редко встречаются ситуации, когда наши действия зависят только от одного условия. Например, можно пойти гулять несмотря на дождь, если на улице достаточно тепло и нет ветра, а также есть зонт и резиновые сапоги.

Чтобы проверить несколько условий, в программировании используются **логические операторы И и ИЛИ**. Разберём подробно, как с их помощью строить в коде логические выражения.

Логические И

Представьте, вы решили завести хомяка и пришли за ним в зоомагазин. У вас есть ряд требований: хомяк должен быть рыжего цвета, не старше двух месяцев, и должен весить больше 100 грамм. Выбирая питомца, вы проверяете эти условия по порядку:



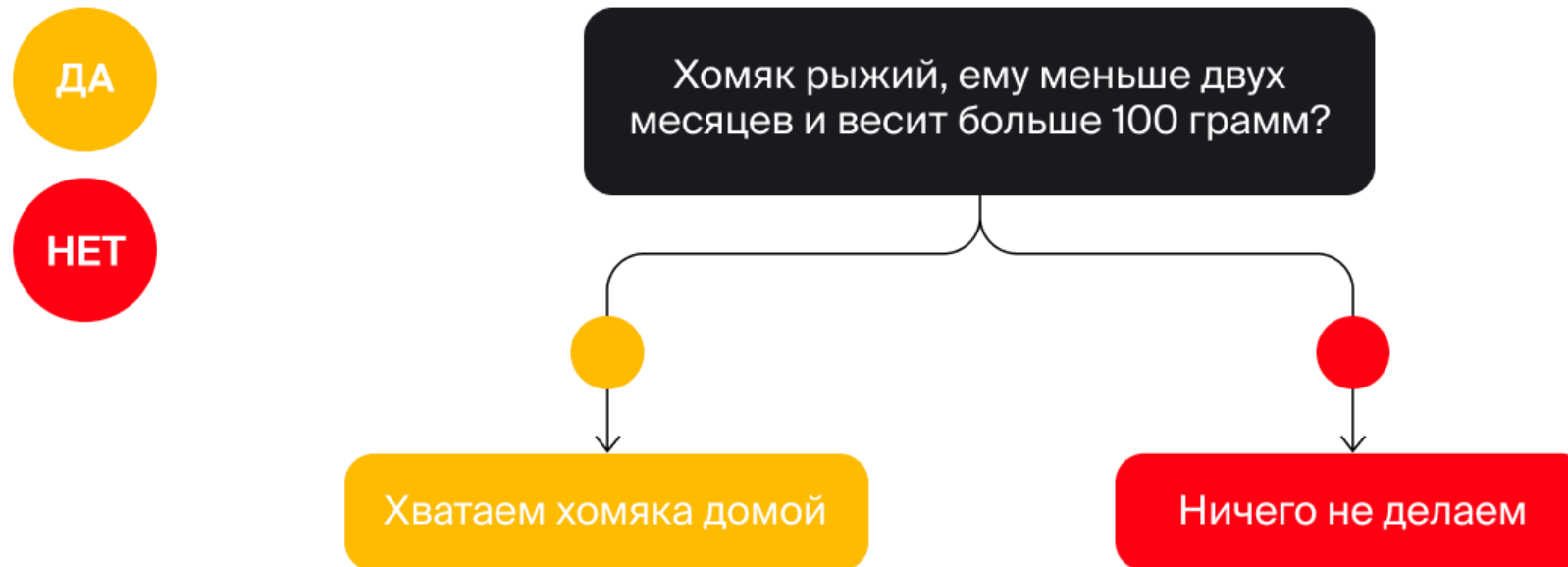



Такая логическая последовательность приведёт только к двум результатам. Если выполнятся все условия — у вас появится хомяк, если не выполнится хотя бы одно — придётся поискать питомца в другом зоомагазине. Чтобы отразить это в коде, нужно одно за другим проверить каждое условие. Можно сделать это так:

```
if (color.equals("Рыжий")) {  
    if (age < 2) {  
        if (weight > 100) {  
            System.out.println("Берём хомяка домой!");  
        }  
    }  
}
```

Или можно объединить эти выражения с помощью **логического И**. Эта операция обозначается в коде через `&&`. Выбор хомяка в коде будет выглядеть вот так:

```
if ((color.equals("Рыжий")) && (age < 2) && (weight > 100)) {  
    System.out.println("Берём хомяка домой!");  
}
```





Если хотя бы одно из выражений-аргументов будет ложным — тогда и условие в ветвлении будет ложным. Если все операнды истинны, то условие тоже истинно.

Часто с помощью `&&` проверяют, входит ли число в диапазон. Например, чтобы убедиться, что пользователь ввёл пароль не меньше заданного количества символов:


```
public class Practice {  
    public static void main(String[] args) {  
        int passwordSize = 7;  
        System.out.println((passwordSize > 5) && (passwordSize < 10));  
    }  
}
```



Результат

true

Если изменить значение переменной `passwordSize` на 20, то программа напечатает **false**, так как условие выражения `passwordSize < 10` будет ложным. Попробуйте подставить в `passwordSize` разные значения и посмотрите, как изменится результат.



При объединении условий с помощью `&&` каждое выражение лучше заключить в скобки. Так код проще читать и можно быть уверенным, что порядок выполнения операций не будет нарушен. Кроме того, не стоит строить длинное логическое выражение напрямую внутри ветвления. Удобнее предварительно сохранить его в переменную типа **boolean**:

```
boolean isHamsterAcceptable = (color.equals("Рыжий")) && (age < 2) &&
(weight > 100);

if (isHamsterAcceptable) {
    System.out.println("Берём хомяка домой!");
}
```

Выберите выражения, где у переменной **answer** будет значение **true**.

A.

```
int x = 0;
boolean answer = (x != 0) && (x > 5) && (x > 2);
```

B.

```
int x = 0;
int y = 5;
boolean answer = (y > x) && (x <= 0) && ((x + y) < 14);
```

C.

```
int a = 15;
boolean answer = (a > 15) && (a >= 0) && (a < 100);
```

D.

```
int a = 11;
int b = a % 2;
boolean answer = (b != 0) && (a >= 0) && (a < 100);
```

E.

```
boolean first = true;
boolean second = "Время".equals("Деньги");
boolean answer = first && second;
```

F.

```
boolean a = false;
int b = 10;
String c = "РазДва";
boolean answer = (!a) && (b <= 10) && (c.equals("Раз" + "Два"));
```

ОТВЕТ

B.

```
int x = 0;
int y = 5;
boolean answer = (y > x) && (x <= 0) && ((x + y) < 14);
```

D.

```
int a = 11;
int b = a % 2;
boolean answer = (b != 0) && (a >= 0) && (a < 100);
```

F.

```
boolean a = false;
int b = 10;
String c = "РазДва";
boolean answer = (!a) && (b <= 10) && (c.equals("Раз" + "Два"));
```




Логические ИЛИ

Допустим, вы твёрдо намерены завести хомяка и не хотите идти в другой магазин. Вы готовы взять питомца, который соответствует хотя бы одному из требований. Это значит, что хомяк должен быть или рыжим, или младше двух месяцев, или весить больше 100 грамм.





Если записывать все условия по порядку, то код будет выглядеть так:

```
if (color.equals("Рыжий")) {  
    System.out.println("Берём хомяка домой!");  
} else {  
    if (age < 2) {  
        System.out.println("Берём хомяка домой!");  
    } else {  
        if (weight > 100) {  
            System.out.println("Берём хомяка домой!");  
        }  
    }  
}
```



Логические ИЛИ

Удобнее объединить все выражения в одно с помощью **логического ИЛИ**. Эта операция возвращает **true**, если выполняется хотя бы один из операндов. Она обозначается в коде через `||`.

Выражение, записанное с помощью логического ИЛИ можно также сохранять в булеву переменную. А для записи его частей-операнд лучше использовать скобки.



Например, значение переменной **strangeLogic** будет **true**, потому что одно из четырёх выражений, объединённых с помощью оператора **||**, является истинным:

```
int a = 1;  
boolean strangeLogic = (a == 0) || (a == 1) || (a == 2) || (a == 3);
```



Результат

true



Переменная станет равна **false**, только если же все его элементы будут ложными:

```
int a = 10;  
boolean strangeLogic = (a == 0) || (a == 1) || (a == 2) || (a == 3);
```



Результат

false



Условие для покупки хомяка с оператором || станет таким.

```
boolean isHamsterAcceptable = (color.equals("Рыжий")) || (age < 2) || (weight > 100);  
  
if (isHamsterAcceptable) {  
    System.out.println("Берём хомяка домой!");  
}
```

Операции логических И и ИЛИ иначе иногда называют логическими умножением и сложением. И вот почему. Если принять, что значение **true** — это 1, а значение **false** — это 0, то общий результат вычислить достаточно легко. Умножение на ноль всегда даст ноль, а если хотя бы одно из слагаемых равно единице, то сумма уже не может быть нулевой.

Первая переменная	Вторая переменная	Результат "ИЛИ"	Результат "И"
true	true	true	true
true	false	true	false
false	true	true	false
false	false	false	false

Какие выражения возвращают **false**.

A.

```
int a = 15;
boolean answer = (a > 15) || (a >= 0) || (a < 100);
```

B.

```
int a = 15;
boolean answer = (a > 15) && (a >= 0) && (a < 100);
```

C.

```
int x = 10 * 10 * 10;
int x++;
boolean isBig = x > 1000;
boolean isNegative = x < 0;
boolean answer = (isBig) || (isNegative);
```

D.

```
int a = 15;
int b = a / 10;
boolean answer = (b == 0) && (a == 0);
```

E.

```
int a = 15;
int b = a;
int c = b;
boolean answer = (c != 15) || (b != 15);
```

ОТВЕТ

B.

```
int a = 15;  
boolean answer = (a > 15) && (a >= 0) && (a < 100);
```

D.

```
int a = 15;  
int b = a / 10;  
boolean answer = (b == 0) && (a == 0);
```

E.

```
int a = 15;  
int b = a;  
int c = b;  
boolean answer = (c != 15) || (b != 15);
```



Порядок логических операций



Порядок логических операций

При написании логического выражения разные операции можно комбинировать друг с другом. При этом важно учитывать порядок их выполнения.

Допустим, что поиски домашнего питомца оказались успешными — в зоомагазине всё-таки нашлись трое подходящих хомяков:


1. Байт. Рыжий короткошёрстный хомяк, ему три месяца, и он упитанный — весит больше 100 грамм.
2. Ниндзя. Рыжий хомяк с белым носом и брюхом, младше двух месяцев, щекастый (больше 100 грамм) и пушистый.
3. Паскаль. Рыжий в чёрную крапинку, тоже младше двух месяцев. Маленький (чуть меньше 100 грамм) и короткошёрстный.



Паскаль
< 2 мес.

Байт
3 мес.

Ниндзя
< 2 мес.



Условие выбора питомца требуется усложнить. Нужно, чтобы хомяк был рыжим или рыже-белым, но не в чёрную крапинку, строго младше двух месяцев, а также либо пушистым, либо упитанным. Чтобы проверить эти пункты, требуется объединить в одно выражение все три операции — логические И и ИЛИ, и отрицание.

```
// логическое И
color.equals("Рыжий") && (age < 2);
// отрицание
!color.equals("В крапинку");
// логическое ИЛИ
weight > 100 || isFluffy == true
```




Порядок логических операций


При комбинировании разных логических операций в одном выражении или условии нужно учитывать приоритет их выполнения в коде:

1. Сначала всегда выполняется отрицание !.
2. Логическое умножение предшествует сложению — поэтому логическое И (&&) в приоритете.
3. Логическое ИЛИ || при наличии других операций выполняется последним.



Порядок логических операций

Логические операции так же, как и арифметические, выполняются слева направо. Операции в скобках вычисляются в первую очередь — поэтому скобки всегда помогут добиться нужного порядка действий в выражении.



Исходя из этих правил, составим логическое выражение для определения подходящего цвета хомяка:

```
boolean isColor = (color.equals("Рыжий") || color.equals("Рыже-белый")) && !color.equals("В крапинку");
```

Хомяк должен быть рыжий или рыже-белый и не должен быть в крапинку. Сначала выполнится отрицание — 1, потом ИЛИ в скобках — 2, и потом И — 3. Выражение с отрицанием `!pattern.equals("В крапинку")` необязательно брать в скобки — оно и так выполнится первым. А вот если не поставить скобки для `||`, то первым выполнится `&&` — результат выражения получится неверным.



Больше ветвлений: if - else if

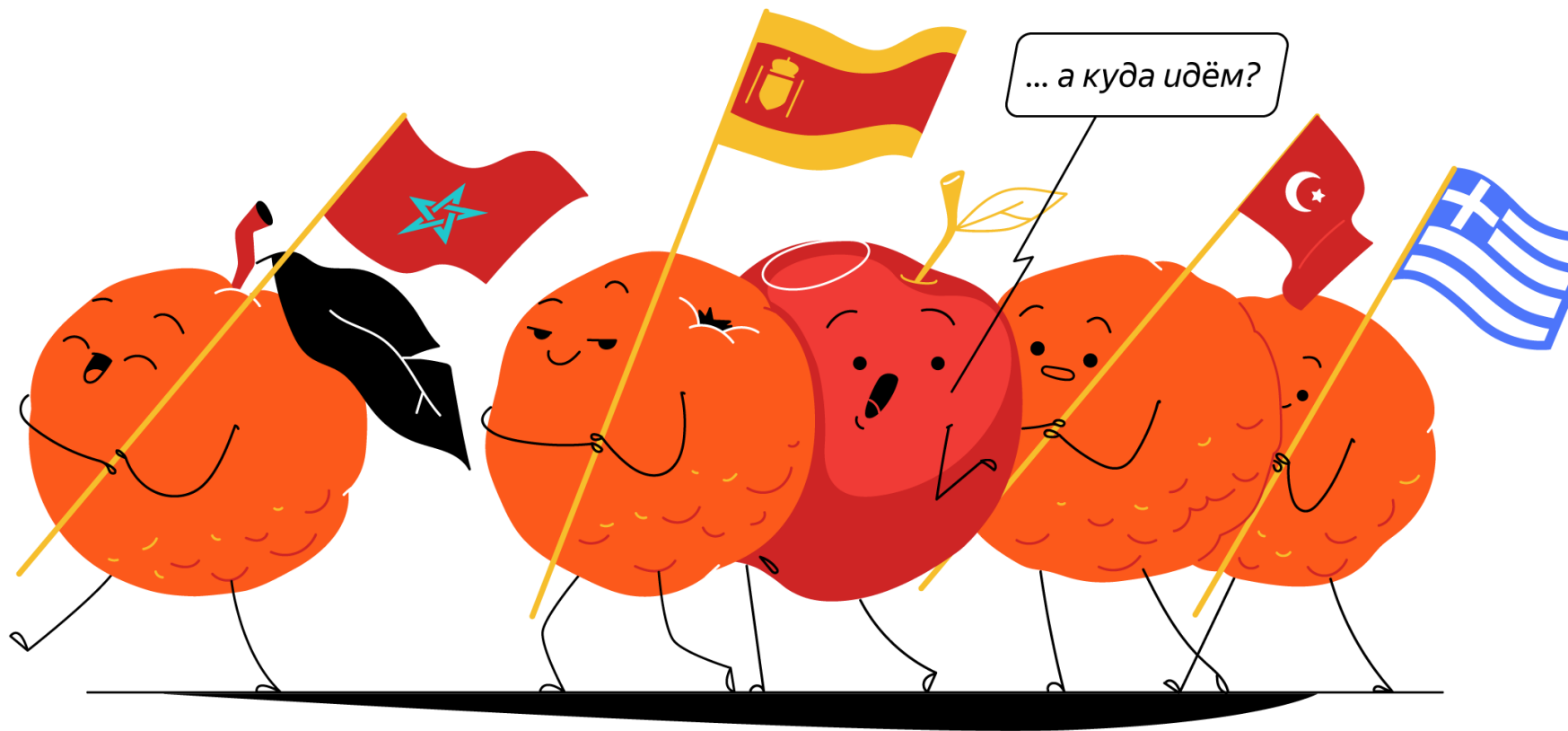


Больше ветвлений: if - else if

Вы научились писать код с условиями с помощью конструкции **if-else**. Теперь ваша программа может выполнить один фрагмент кода, если условие выполняется, и другой — если нет. Но ведь в жизни обычно вариантов выбора не два, а больше. В примере с рынком условие было таким: если яблоки красные, то покупаем их, иначе покупаем апельсины.

Больше ветвлений: if - else if

Предположим, что для апельсинов тоже существует условие. Например, чтобы они обязательно были из Марокко. Тогда алгоритм ваших действий будет таким: если есть красные яблоки, то купить их, иначе, если апельсины из Марокко, то покупаем их, иначе просто идём домой.





В коде эту логику тоже можно реализовать:

```
if (условие 1) {  
    код #1  
} else if (условие 2) {  
    код #2  
} else {  
    код #3  
}  
код #4
```

- Если условие 1 истинно, тогда выполнится только код #1.
- Когда условие 1 ложно и условие 2 истинно, то сработает только код #2.
- Если оба условия ложны, сработает только код #3.

Вопрос

Допустим, вы пришли на рынок. Яблоки остались только зелёные, а апельсины — исключительно испанские. Что вы сделаете?

```
if (яблоки красные) {  
    купить яблоки;  
} else if (апельсины из Марокко) {  
    купить апельсины;  
} else {  
    тяжело вздохнуть;  
}  
пойти домой;
```

- A) Пойду домой.
- B) Куплю яблоки и пойду домой.
- C) Наберу яблок, апельсинов и отправлюсь домой.
- D) Тяжело вздохну и пойду домой



Ответ

- A) Пойду домой.
- B) Куплю яблоки и пойду домой.
- C) Наберу яблок, апельсинов и отправлюсь домой.
- D) Тяжело вздохну и пойду домой

Добавляем if - else if в финансовое приложение

С логикой финансового приложения проблема очень похожая. Ваш цифровой советник уже умеет конвертировать тенге в евро и доллары. А что, если вы захотите добавить ещё одну валюту, например японскую иену? Программа начнёт работать неправильно и вместо иен напечатает значение в евро:

```
if (currency.equals("USD")) {  
    System.out.println("Ваши сбережения в долларах: " + tenges / rateUSD);  
} else {  
    System.out.println("Ваши сбережения в евро: " + tenges / rateEUR);  
}
```



Добавляем if - else if в финансовое приложение

Исправим финансовое приложение так, чтобы оно научилось работать с иенами. Для этого после **else** нужно добавить ещё одну проверку имени валюты, в которую нужно конвертировать.

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

        double tenges = 124356.5;
        double rateUSD = 444.06;
        double rateEUR = 489.32;
        double rateJPY = 3.12; // Курс японской иены

        String currency = "EN";
        System.out.println("Вы конвертируете тенге в " + currency);

        if (currency.equals("USD")) {
            System.out.println("Ваши сбережения в долларах: " + tenges / rateUSD);
        } else if (currency.equals("EUR")) { // Проверяем имя валюты
            System.out.println("Ваши сбережения в евро: " + tenges / rateEUR);
        } else {
            System.out.println("Ваши сбережения в иенах: " + tenges / rateJPY);
        }
    }
}
```



Больше ветвлений: if - else if

Так можно писать ветвления любого размера: ограничений нет. Но будьте осторожны — большие ветвления сложно читать! Попробуйте поменять значение переменной **currency** так, чтобы произошла конвертация в иены.



Задача 1

Добавьте в финансовое приложение ещё один совет. Если до зарплаты осталось больше 50_000 и при этом меньше 150_000 тенге, пусть выводится такая фраза: «Неплохо! Прикупите долларов и зайдите поужинать в классное место.».

Задача 1

```
public class Practice {  
    public static void main(String[] args) {  
        double moneyBeforeSalary = 15000.0; // Количество денег до зарплаты  
  
        if (moneyBeforeSalary < 15_000) {  
            System.out.println("Сегодня лучше поесть дома. Экономьте, и вы дотянете до зарплаты!");  
        } else if (moneyBeforeSalary < 50_000) {  
            System.out.println("Окей, пора в Макдак!");  
        } ... {  
            System.out.println("Неплохо! Прикупите долларов и зайдите поужинать в классное место.");  
        } ... {  
            System.out.println("Класс! Заказывайте крабов!");  
        }  
    }  
}
```

Решение

```
public class Practice {  
    public static void main(String[] args) {  
        double moneyBeforeSalary = 15000.0; // Количество денег до зарплаты  
  
        if (moneyBeforeSalary < 15_000) {  
            System.out.println("Сегодня лучше поесть дома. Экономьте, и вы дотянете до зарплаты!");  
        } else if (moneyBeforeSalary < 50_000) {  
            System.out.println("Окей, пора в Макдак!");  
        } else if (moneyBeforeSalary < 150_000) {  
            System.out.println("Неплохо! Прикупите долларов и зайдите поужинать в классное место.");  
        } else {  
            System.out.println("Класс! Заказывайте крабов!");  
        }  
    }  
}
```




Задача 2

Как мы ни старались, но в код прокралась ошибка! Вам предстоит её исправить: конвертация в иены работает, но проблема возникает, если попытаться конвертировать в новую валюту, например, датскую крону — **DKK**. Программа в таком случае конвертирует в иены, а хотелось бы получить сообщение об ошибке, ведь приложение пока не поддерживает такую валюту.

```
public class Practice {  
    public static void main(String[] args) {  
  
        double tenges = 124356.5;  
        double rateUSD = 444.06;  
        double rateEUR = 489.32;  
        double rateJPY = 3.12; // Курс японской иены  
  
        String currency = "DKK";  
        System.out.println("Вы конвертируете тенге в " + currency);  
  
        if (currency.equals("USD")) {  
            System.out.println("Ваши сбережения в долларах: " + tenges / rateUSD);  
        } else if (currency.equals("EUR")) {  
            System.out.println("Ваши сбережения в евро: " + tenges / rateEUR);  
        } ... {  
            System.out.println("Ваши сбережения в иенах: " + tenges / rateJPY);  
        } ... {  
            System.out.println("Валюта не поддерживается.");  
        }  
    }  
}
```



Решение

```
public class Practice {  
    public static void main(String[] args) {  
  
        double tenges = 124356.5;  
        double rateUSD = 444.06;  
        double rateEUR = 489.32;  
        double rateJPY = 3.12; // Курс японской иены  
  
        String currency = "DKK";  
        System.out.println("Вы конвертируете тенге в " + currency);  
  
        if (currency.equals("USD")) {  
            System.out.println("Ваши сбережения в долларах: " + tenges / rateUSD);  
        } else if (currency.equals("EUR")) {  
            System.out.println("Ваши сбережения в евро: " + tenges / rateEUR);  
        } else if (currency.equals("JPY")) {  
            System.out.println("Ваши сбережения в иенах: " + tenges / rateJPY);  
        } else {  
            System.out.println("Валюта не поддерживается.");  
        }  
    }  
}
```



Вложенные условия



Вложенные условия

Вы научились применять условные выражения в коде, чтобы сделать выбор из нескольких разных вариантов: например, купить красные яблоки, а если их нет, то марокканские апельсины. Однако и этого может быть недостаточно.

Вложенные условия

Например, вы хотите купить три килограмма яблок, но тут вспоминаете, что дома вроде бы ещё оставались фрукты, а значит, вам хватит и одного кило румяных «Фуджи». Описать эти условия в коде можно следующим образом:

```
if (яблоки красные) {  
    // Вложенный if  
    if (дома есть фрукты) {  
        купить 1 кг яблок;  
    } else {  
        купить 3 кг яблок;  
    }  
} else {  
    купить апельсины;  
}
```

Такое ветвление называется **вложенным if-else**: внутри одного блока с кодом находится другой. Внутри вложенного **if** вы можете написать сколько угодно условных выражений. Но старайтесь не увлекаться вложенностью — ваш код может стать сложным для понимания.



В большинстве стран мира водить машину разрешается с 18 лет. В Америке сесть за руль можно раньше, но это зависит от штата. Что будет напечатано в результате выполнения кода?

```
int age = 17;
String country = "USA";
String state = "Alaska";

if (age < 18) {
    if (country.equals("USA")) {
        if (state.equals("New York")) {
            System.out.println("Можете водить машину, но вам нельзя покидать пределы Нью-Йорка.");
        } else {
            System.out.println("Придётся подождать до 18 или переехать в Нью-Йорк.");
        }
    } else {
        System.out.println("Придётся подождать до 18.");
    }
} else {
    System.out.println("Можете водить где угодно.");
}
```



Вопрос

- A) Вы можете водить машину, но вам нельзя покидать пределы Нью-Йорка.
- B) Придется подождать до 18 или переехать в Нью-Йорк.
- C) Придется подождать до 18.
- D) Можете водить где угодно.



Ответ

В) Придется подождать до 18 или переехать в Нью-Йорк.


Добавляем вложенные условия в финансовое приложение

Теперь вы умеете писать вложенные условия и сможете улучшить свою программу-консультант. В прошлом уроке вы уже научили её давать финансовые советы, а теперь постараемся максимально приблизить их к реальности. Сейчас мы учитываем только остаток денег на счёте, но не принимаем во внимание количество дней до зарплаты. Если у вас 150_000 тенге и перевод придёт уже завтра, то проблем никаких, можно и в ресторан сходить. А если при тех же 150_000 тенге ждать средств придётся ещё месяц? Скорее всего, не стоит в такой момент есть крабов. Было бы классно учитывать это, давая советы.

Задача

Исправьте код программы-советчика. Если до зарплаты остаётся меньше 10 дней, сообщения останутся прежними, если же больше 10 — пускай печатается совет для предыдущей категории. Например, если на счету 150_000 тенге и при этом до зарплаты остаётся 17 дней, то напечатается не «Неплохо! Прикупите долларов и зайдите поужинать в классное место.», а «Окей, пора в Макдак!».

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



Импорт пакетов и монтаж приложения



Импорт пакетов и монтаж приложения

Теперь можно объединить конвертер валют и финансового советника в одно приложение. В программе будет две команды: *convert* (англ. «конвертировать») и *advice* (англ. «совет»). Пользователь приложения сможет ввести команду и дополнительные параметры (сумму на счёте и количество дней до зарплаты), а в ответ, в зависимости от выбранной опции, получит результат конвертации или совет относительно ужина.

Теперь можно объединить конвертер валют и финансового советника в одно приложение. В программе будет две команды: *convert* (англ. «конвертировать») и *advice* (англ. «совет»). Пользователь приложения сможет ввести команду и дополнительные параметры (сумму на счёте и количество дней до зарплаты), а в ответ, в зависимости от выбранной опции, получит результат конвертации или совет относительно ужина.

```
import java.util.Scanner; // Импортировали пакет

public class Practice {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in); // Объявили переменную с типом Scanner
        String command = scanner.nextLine(); // Считали строку из консоли

        System.out.println("Вы ввели команду " + command);
    }
}
```


В методе `main` сначала создаётся переменная с типом `Scanner` (англ. «считывать») и именем `scanner`. Тип пишется с заглавной буквы, а имя переменной — со строчной. После объявления ей присваивается значение (`new Scanner(System.in)`).

```
Scanner scanner = new Scanner(System.in);
```

The diagram illustrates the components of the code line `Scanner scanner = new Scanner(System.in);`. It features three horizontal lines with vertical lines extending downwards to descriptive text:

- Scanner**: Тип переменной (Type of variable)
- scanner**: Имя переменной (Variable name)
- = new Scanner(System.in);**: Создаёт объект типа Scanner, считывающий из потока ввода System.in (Creates an object of type Scanner, reading from the input stream System.in)



Импорт пакетов и монтаж приложения

До этого мы рассматривали только простые типы — `int`, `double`, `boolean`. Они называются **примитивными**. А есть «сложные» типы, значения которых — это некий **объект**, умеющий выполнять разные действия. Для создания объектов всегда используется слово `new` (англ. «новый»). Переменная типа `Scanner` умеет считывать значения из потока ввода — `System.in`. Это указывается при её создании.

На следующей строке у переменной с именем **scanner** вызывается метод **nextLine()** (англ. «следующий»). Для этого после имени переменной ставится точка, и после неё — имя метода. **nextLine()** считывает из терминала строку и сохраняет её в переменную с именем **command**.

```
String command = scanner.nextLine();
```

- Переменная, в которую считывается ввод пользователя
- Имя переменной, у которой вызываем команду
- Команда считывает ввод пользователя

Импорт пакетов и монтаж приложения

💡 У типа **Scanner** есть ещё много разных команд. Например, `nextInt()` для считывания целого числа и `nextDouble()` — дробного. Ещё одна команда `next()` позволяет считывать не строку целиком, а первое слово до пробела. Такие готовые типы позволяют собирать программу из фрагментов, как конструктор. Вместо того чтобы описывать сложную логику для считывания символов из консоли, вы просто используете тип **Scanner**.



Импорт пакетов и монтаж приложения

В Java есть очень много готовых типов. Для удобства они хранятся в разных папках, которые называются **пакетами**. Это очень похоже на то, как файлы структурированы на жёстком диске компьютера: фотографии хранятся в папке с названием «Фото», а аудиофайлы — в «Музыке». Подобным образом организуют и код, только не по папкам, а по пакетам.

Импорт пакетов и монтаж приложения

Если тип хранится в каком-то пакете, то для того, чтобы его использовать, этот пакет сначала нужно **импортировать** (англ. *import*). Тип **Scanner** хранится в пакете с названием **java.util**, и для того, чтобы использовать его в коде, нужно написать следующую команду:

```
import java.util.Scanner;
```

Теперь вы сможете использовать тип **Scanner** и все его команды. Если попытаться запустить программу без **import**, произойдёт ошибка. Попробуйте запустить код ниже и убедитесь в этом.

```
// Не импортировали пакет java.util.Scanner;  
public class Practice {  
    public static void main(String[] args) {  
  
        Scanner scanner = new Scanner(System.in); // Объявили переменную с типом Scanner  
        String command = scanner.nextLine(); // Считали строку из консоли  
  
        System.out.println("Вы ввели команду " + command);  
    }  
}
```

Результат

```
src/Practice.java:5:9
java: cannot find symbol
  symbol:   class Scanner
  location: class Practice
```

```
src/Practice.java:5:31
java: cannot find symbol
  symbol:   class Scanner
  location: class Practice
```

Java не знает, где искать **Scanner**, и сообщает нам об этом в сообщении об ошибке— **java: cannot find symbol** (англ. «не могу найти символ»). Если вернуть в код **import**, проблема исчезнет.

Задача 1

Это последний урок темы и вы сможете применить все знания в финальной версии приложения для финансов. В нём будет две команды: **advice** и **convert**. Мы написали заготовку для обработки команды **convert** — она нужна для конвертации валюты. Пока только в доллары, но в следующем задании вы это исправите, а пока — допишите обработку команды **advice**.

Ещё одно важное изменение: теперь пользователь будет вводить количество денег на счёте и дней до зарплаты. Эти значения больше не будут храниться в переменной, они будут считываться с помощью типа **Scanner**. Если пользователь ошибётся и введёт какую-то другую команду, то пусть появляется сообщение «Извините, такой команды пока нет».

https://github.com/practicetasks/java_tasks/blob/main/conditions/task_2/README.md



Решение

<https://gist.github.com/practicetasks/9de05170bbf094ae8a4d8eabe2756223>

Задача 2

Остался последний штрих! Приложение пока что умеет конвертировать тенге только в одну валюту — доллары. Именно значение **USD** хранится в переменной **currency**.

https://github.com/practicetasks/java_tasks/blob/main/conditions/task_3/README.md



Решение

<https://gist.github.com/practicetasks/b073391a2c229d41d76cfa7aececb586>