**Содержание урока**

Презентация ([скачать](https://stepik.org/media/attachments/lesson/12759/module2.pdf))

1. [О чем этот модуль](https://stepik.org/lesson/%D0%9F%D1%80%D0%B8%D0%BC%D0%B8%D1%82%D0%B8%D0%B2%D0%BD%D1%8B%D0%B5-%D1%82%D0%B8%D0%BF%D1%8B-12759/step/2?course=Java-%D0%91%D0%B0%D0%B7%D0%BE%D0%B2%D1%8B%D0%B9-%D0%BA%D1%83%D1%80%D1%81&unit=3107)
2. [Примитивные и ссылочные типы](https://stepik.org/lesson/%D0%9F%D1%80%D0%B8%D0%BC%D0%B8%D1%82%D0%B8%D0%B2%D0%BD%D1%8B%D0%B5-%D1%82%D0%B8%D0%BF%D1%8B-12759/step/3?course=Java-%D0%91%D0%B0%D0%B7%D0%BE%D0%B2%D1%8B%D0%B9-%D0%BA%D1%83%D1%80%D1%81&unit=3107)
3. [Тип boolean](https://stepik.org/lesson/%D0%9F%D1%80%D0%B8%D0%BC%D0%B8%D1%82%D0%B8%D0%B2%D0%BD%D1%8B%D0%B5-%D1%82%D0%B8%D0%BF%D1%8B-12759/step/4?course=Java-%D0%91%D0%B0%D0%B7%D0%BE%D0%B2%D1%8B%D0%B9-%D0%BA%D1%83%D1%80%D1%81&unit=3107)
4. [Контрольный вопрос](https://stepik.org/lesson/%D0%9F%D1%80%D0%B8%D0%BC%D0%B8%D1%82%D0%B8%D0%B2%D0%BD%D1%8B%D0%B5-%D1%82%D0%B8%D0%BF%D1%8B-12759/step/5?course=Java-%D0%91%D0%B0%D0%B7%D0%BE%D0%B2%D1%8B%D0%B9-%D0%BA%D1%83%D1%80%D1%81&unit=3107)
5. [Задача](https://stepik.org/lesson/%D0%9F%D1%80%D0%B8%D0%BC%D0%B8%D1%82%D0%B8%D0%B2%D0%BD%D1%8B%D0%B5-%D1%82%D0%B8%D0%BF%D1%8B-12759/step/6?course=Java-%D0%91%D0%B0%D0%B7%D0%BE%D0%B2%D1%8B%D0%B9-%D0%BA%D1%83%D1%80%D1%81&unit=3107)
6. [Целочисленные типы](https://stepik.org/lesson/%D0%9F%D1%80%D0%B8%D0%BC%D0%B8%D1%82%D0%B8%D0%B2%D0%BD%D1%8B%D0%B5-%D1%82%D0%B8%D0%BF%D1%8B-12759/step/7?course=Java-%D0%91%D0%B0%D0%B7%D0%BE%D0%B2%D1%8B%D0%B9-%D0%BA%D1%83%D1%80%D1%81&unit=3107)
7. [Задача](https://stepik.org/lesson/%D0%9F%D1%80%D0%B8%D0%BC%D0%B8%D1%82%D0%B8%D0%B2%D0%BD%D1%8B%D0%B5-%D1%82%D0%B8%D0%BF%D1%8B-12759/step/8?course=Java-%D0%91%D0%B0%D0%B7%D0%BE%D0%B2%D1%8B%D0%B9-%D0%BA%D1%83%D1%80%D1%81&unit=3107)
8. [Тип char](https://stepik.org/lesson/%D0%9F%D1%80%D0%B8%D0%BC%D0%B8%D1%82%D0%B8%D0%B2%D0%BD%D1%8B%D0%B5-%D1%82%D0%B8%D0%BF%D1%8B-12759/step/9?course=Java-%D0%91%D0%B0%D0%B7%D0%BE%D0%B2%D1%8B%D0%B9-%D0%BA%D1%83%D1%80%D1%81&unit=3107)
9. [Вещественные типы](https://stepik.org/lesson/%D0%9F%D1%80%D0%B8%D0%BC%D0%B8%D1%82%D0%B8%D0%B2%D0%BD%D1%8B%D0%B5-%D1%82%D0%B8%D0%BF%D1%8B-12759/step/10?course=Java-%D0%91%D0%B0%D0%B7%D0%BE%D0%B2%D1%8B%D0%B9-%D0%BA%D1%83%D1%80%D1%81&unit=3107)
10. [Полезные классы: Math, BigInteger, BigDecimal](https://stepik.org/lesson/%D0%9F%D1%80%D0%B8%D0%BC%D0%B8%D1%82%D0%B8%D0%B2%D0%BD%D1%8B%D0%B5-%D1%82%D0%B8%D0%BF%D1%8B-12759/step/11?course=Java-%D0%91%D0%B0%D0%B7%D0%BE%D0%B2%D1%8B%D0%B9-%D0%BA%D1%83%D1%80%D1%81&unit=3107)
11. [Задача](https://stepik.org/lesson/%D0%9F%D1%80%D0%B8%D0%BC%D0%B8%D1%82%D0%B8%D0%B2%D0%BD%D1%8B%D0%B5-%D1%82%D0%B8%D0%BF%D1%8B-12759/step/12?course=Java-%D0%91%D0%B0%D0%B7%D0%BE%D0%B2%D1%8B%D0%B9-%D0%BA%D1%83%D1%80%D1%81&unit=3107)
12. [Вопрос](https://stepik.org/lesson/%D0%9F%D1%80%D0%B8%D0%BC%D0%B8%D1%82%D0%B8%D0%B2%D0%BD%D1%8B%D0%B5-%D1%82%D0%B8%D0%BF%D1%8B-12759/step/13?course=Java-%D0%91%D0%B0%D0%B7%D0%BE%D0%B2%D1%8B%D0%B9-%D0%BA%D1%83%D1%80%D1%81&unit=3107)
13. [Вопрос](https://stepik.org/lesson/%D0%9F%D1%80%D0%B8%D0%BC%D0%B8%D1%82%D0%B8%D0%B2%D0%BD%D1%8B%D0%B5-%D1%82%D0%B8%D0%BF%D1%8B-12759/step/14?course=Java-%D0%91%D0%B0%D0%B7%D0%BE%D0%B2%D1%8B%D0%B9-%D0%BA%D1%83%D1%80%D1%81&unit=3107)
14. [Задача](https://stepik.org/lesson/%D0%9F%D1%80%D0%B8%D0%BC%D0%B8%D1%82%D0%B8%D0%B2%D0%BD%D1%8B%D0%B5-%D1%82%D0%B8%D0%BF%D1%8B-12759/step/15?course=Java-%D0%91%D0%B0%D0%B7%D0%BE%D0%B2%D1%8B%D0%B9-%D0%BA%D1%83%D1%80%D1%81&unit=3107)
15. **&** - Бинарный оператор ***логическое "И"****.*Результатом выражения вида **A&B** является **true (правда)**, если значение обоих операндов **A и B**равны**true(правда).**  Если хотя бы один из операндов имеет значение**false (ложь)**, то результатом выражения **A&B** является значение **false (ложь)**​​​​​​​​​​​​​
16. **​​​&&** - Бинарный оператор ***логическое "И"***. Вычисляемый по *упрощенной*схеме. Результатом выражения **A&&B** такой же, как и выражение**A&B**но в случае с оператором **&&**,  если при проверке значения операнда**А** оказывается, что оно равно **false (ложь)**, то значение операнды В уже не вычисляется, а сразу результатом  выражения **А&&В** возвращается значение **false (ложь).**
17. **|**- Бинарный оператор***логическое "ИЛИ"***, Результатом выражения вида**А|В** является **true (правда)**, если значение хотя бы одной из операндов**А** или**В** равно**true (правда)**. Если значение обеих операндов равны **false (ложь)**, то результатом выражения**А|В**возвращается значение **false (ложь).**
18. **||**- Бинарный оператор***логическое* "ИЛИ"**, который вычисляется по упрощенной схеме. Результатом выражения **А||В**, совпадает со значением выражения**А|В**, но в случае с оператором ||, если при проверки значения операнды**А**, оказывается, что она равна**true (правда)**, то значение операнды **В** уже не вычисляется, а сразу результатом выражения **А||B** возвращается значение **true (правда)**.
19. **^** - Бинарный оператор***логическое* "ИСКЛЮЧАЮЩЕЕ ИЛИ"**. Результатом выражения вида **A^B** является**true (правда)**, если один операнд имеет значение **true (правда)**, а другой операнд имеет значение **false (ложь)**. Если значение обоих операндов одновременно равны**false (ложь)**, или  если их значения одновременно ровны **true (правда)**, результатом выражения **А^B** возвращается значение **false (ложь)**.
20. **!** - Унарный оператор ***логического* "отрицания"**. Результатом выражения вида **!А** равняется значение **true (правда)**, если значение операнды **А** равно **false (ложь)**, если значение операнды **А**равно **true (правда)**, то результатом выражения   **!А** является значение **false (ложь).**
21. | -or, || - or, ^-xor, !=-not equal, ==-equal, &-and, теперь на "пальцах", |(или) и ||(или) оба или, они дают один и тот же результат
22. ^(исключающее или) и !=(не равно) оба основываются на логическом "отрицании" и дают один и тот же результат.
23. |(или) и !=(не равно) если 1+1 |(или) 2+2 и т.д. т.е если что либо истино, != если 1+1 !=(не равно) 2+2 и т.д. т.е. что либо ложно, (одно основывается на истине, другая на лжи) операторы дадут разный результат
24. 1+1 &(и) 2+2 если первый операнд ложь то второй не учитывается, 1+1 ==(равно) 2+2 оба операнда учитываются . Операторы дадут разный результат.
25. Все понятно?:)

Реализуйте метод, возвращающий true, если среди четырех его аргументов ровно два истинны (любые). Во всех остальных случаях метод должен возвращать false.  
  
Воспользуйтесь шаблоном кода, который предлагает система. Ввод-вывод будет сделан за вас. Вам надо только проанализировать переданные в метод booleanExpression значения (a, b, c, d) и вернуть результат. Попробуйте составить формулу с использованием булевых операторов. Если не получается, вернитесь к этому заданию после просмотра степов про условные операторы и циклы.  
  
При записи сложных выражений рекомендуется использовать скобки, чтобы не запутаться в порядке применения операторов.  
  
В качестве примера уже указано заведомо некорректное решение задачи. Исправьте его.  
  
Совет тем, у кого не проходит какой-то из тестов. В данной задаче возможно всего 16 комбинаций значений входных параметров. Их можно выписать на бумажку, посчитать для них правильные ответы и сравнить с тем, что выдает ваше решение. Попробуйте самостоятельно проделать это, найти ошибку и исправить решение.

public static boolean booleanExpression(boolean a, boolean b, boolean c, boolean d) {

return ((!a) & (!b) & (c) & (d)) ^ ((a) & (b) & (!c) & (!d)) ^ ((!a) & (b) & (c) & (!d)) ^ ((a) & (!b) & (!c) & (d)) ^ ((!a) & (b) & (!c) & (d)) ^ ((a) & (!b) & (c) & (!d));

}

0000  
0001  
0010  
0011  
0100  
0101  
0110  
0111  
1000  
1001  
1010  
1011  
1100  
1101  
1110  
1111

public static boolean booleanExpression(boolean a, boolean b, boolean c, boolean d) { return ((a?1:0) + (b?1:0) + (c?1:0) + (d?1:0)) == 2; }

Какой тип имеет литерал 0x0bp3?

Введите имя соответствующего примитивного типа. Напоминаем, что Java чувствительна к регистру символов.

В **шестнадцатеричной** системе счисления числа записываются с помощью префиксов **0x** или **0X**. Допустимые цифры — от 0 до 15, где числа 10-15 обозначаются символами A-F соответственно.

Что означает **p** в записи  0x0bp3? **Для шестнадцатеричных литералов всегда требуется экспонента и обозначается буквой ASCII p или P, за которой следует опционально знаковое целое число.**

 Рассмотрим литерал 0x0bp3 более подробно:

0b16 = 1110

p3 = 2^323

Получаем, что **0x0bp3** = 11 \* 2^3 = 11 \* 8 = 8811∗23=11∗8=88.

Итак,  0x0bp3 - шестнадцатеричный литерал с плавающей точкой.

Любое число с плавающей точкой создает тип double. Если необходимо использовать тип float, следует добавить в конце символ f или F. В этом случае double будет сведен к типу float.

Желаю успехов!

Реализуйте метод flipBit, изменяющий значение одного бита заданного целого числа на противоположное. Данная задача актуальна, например, при работе с [битовыми полями](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D1%82%D0%BE%D0%B2%D0%BE%D0%B5_%D0%BF%D0%BE%D0%BB%D0%B5" \o "Link: https://ru.wikipedia.org/wiki/Битовое_поле" \t "_blank).

Договоримся, что биты нумеруются от младшего (индекс 1) к старшему (индекс 32).

Воспользуйтесь предоставленным шаблоном. Декларацию класса, метод main и обработку ввода-вывода добавит проверяющая система.

**Sample Input:**

0 1

**Sample Output:**

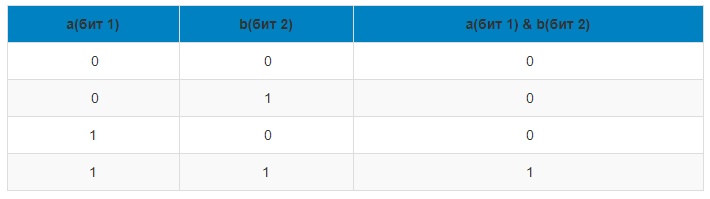
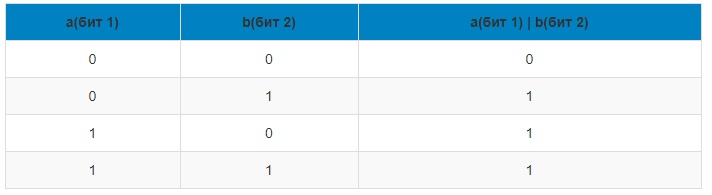
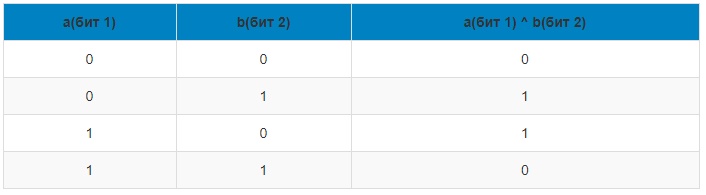
1

public static int flipBit(int value, int bitIndex) {

return value ^ (1 << bitIndex-1); // put your implementation here

}

***Решение по пунктам:***

1. bitIndex-1, т.к. нумерация начинается с нуля.
2. (1 << bitIndex-1) - здесь мы сдвигаем единицу влево на нужное число бит (это число bitIndex-1)
3. value ^ (1 << bitIndex-1) — меняем нужный бит, т.к., например, при 01011011101^0001 получаем 01001011101.
4. Основные логические операции. AND, NOT, OR и XOR (исключающее или)
5. [Новости OTUS](https://otus.ru/nest/otus-news/)
6. Теги: xor, or, and, not, логические операции, исключающее или, бит, битовые операции
7. В этой статье мы поговорим о некоторых битовых операциях. Рассмотрим основные из них: XOR (исключающее ИЛИ), AND (И), NOT (НЕ) а также OR (ИЛИ).
8. Как известно, минимальной единицей измерения информации является **бит**, который хранит одно из 2-х значений: 0 (**False**, ложь) либо 1 (**True**, истина). Таким образом, битовая ячейка может одновременно находиться лишь в одном из двух возможных состояний.
9. Для манипуляций с битами используют определённые операции — **логические или булевые**. Они могут применяться к любому биту, вне зависимости от того, какое у него значение — ноль или единица. Что же, давайте посмотрим на примеры использования трёх основных логических операций.
10. **Логическая операция AND (и)**
11. **AND** обозначается знаком &.
12. Оператор AND выполняется с 2-мя битами, возьмём, к примеру, a и b. Результат выполнения операции AND равен 1, если a и b равняются 1. В остальных случаях результат равен 0. Например, с помощью AND вы можете узнать, чётное число или нет.
13. Посмотрите на таблицу истинности операции AND:
14. 
15. **Логическая операция OR (ИЛИ)**
16. Обозначается знаком |.
17. Оператор **OR** также выполняется с 2-мя битами (a и b). Результат равен 0, если a и b равны 0, иначе он равен 1. Смотрим таблицу истинности.
18. 
19. **Логическая операция XOR (исключающее ИЛИ)**
20. Оператор XOR обозначается ^.
21. **XOR** выполняется с 2-мя битами (a и b). Результат выполнения операции XOR (**исключающее ИЛИ**) равен 1, когда один из битов b или a равен 1. В остальных ситуациях результат применения оператора XOR равен 0.
22. Таблица истинности логической операции для XOR (исключающее ИЛИ) выглядит так:
23. 
24. Используя XOR (исключающее ИЛИ), вы можете поменять значения 2-х переменных одинакового типа данных, не используя временную переменную. А ещё, посредством XOR можно зашифровать текст, например:
25. String msg = "This is a message";
26. char[] message = msg.toCharArray();
27. String key = ".\*)";
28. String encryptedString = new String();
29. for(int i = 0; i< message.length; i++){
30. encryptedString += message[i]^key.toCharArray()[i%key.length()];
31. }
32. Согласен, XOR — далеко не самый надёжный метод шифрования, но это не значит, что его нельзя сделать частью какого-либо шифровального алгоритма.
33. **Логическая операция NOT (НЕ)**
34. Это побитовое отрицание, поэтому выполняется с одним битом и обозначается ~.
35. Результат зависит от состояния бита. Если он в нулевом состоянии, то итог операции — единица и наоборот. Всё предельно просто.
36. 
37. Эти 4 логические операции следует запомнить в первую очередь, т. к. с их помощью можно получить практически любой возможный результат. Также существуют такие операции, как << (побитовый сдвиг влево) и >> (побитовый сдвиг вправо).