Лекция 3. Ветвление. Логический тип

Ветвление

- Часто требуется, чтобы код выполнялся только если выполняется некоторое условие
- Например, если значение делителя равно 0, то выдать ошибку. Иначе – выполнить деление
- Для этого во многих языках есть конструкция **ветвление (условный оператор)**

Высказывания, истина и ложь

- **Логическим высказыванием** называется некоторое предложение, про которое можно сказать **истинно** оно или **ложно**
- Примеры:
  - Сейчас идет дождь
  - Завтра четверг
  - Число x больше 5
- Эти высказывания можно вычислить в конкретный момент времени и получить результат – истина или ложь

Логический тип boolean

- Используется для задания условий
- Имеет два возможных значения – true (истина) и false (ложь)
- Пример:
- ```
  boolean a = true;
  boolean b = false;
  ```
  ```

Логические выражения

Выражение	Символ	Пример
Проверка на равенство	==	a == b
Проверка на неравенство	!=	a != b
Строгое сравнение	> и <	a > b a < b
Нестрогое сравнение	>= и <=	a >= b a <= b

- Пример (предположим, у нас есть Scanner s):

```
double a = s.nextDouble();
System.out.println(a > 5);  // выдаст true, если
// введенное число > 5, иначе - false
```

Не нужно путать проверку на равенство == с оператором присваивания =

Логические связки

Название	Синонимы	Операторы	Примеры
Логическое И	Конъюнкция	&&	a >= 5 && b == 3
Логическое ИЛИ	Дизъюнкция	||	3 < 5 || b > 4
Логическое НЕ	Отрицание	!	!(a == 4)

Логическое И (конъюнкция)

a	b	a && b
false	false	false
false	true	false
true	false	false
true	true	true

- **Конъюнкция** истинна (результат - true) только если истинны все подвыражения (все подвыражения дают true)

- Пример:

```
int a = 5, b = 6;
boolean c = (a < b) && (a < b – 5);    // false
boolean d = (a < b) && (b != a);       // true
```

Логическое ИЛИ (дизъюнкция)

a	b	a || b
false	false	false
false	true	true
true	false	true
true	true	true

- **Дизъюнкция** истинна если истинно хотя бы одно подвыражение

Логическое НЕ (отрицание)

a	!a
false	true
true	false

- Просто обращение значения выражения

Исключающее ИЛИ (XOR)

a	b	a ^ b
false	false	false
false	true	true
true	false	true
true	true	false

- **Исключающее ИЛИ (XOR)** - истинно если истинно ровно одно подвыражение
- Это очень редкая операция, ее почти никогда не применяют
- Мы ее рассматриваем для общего развития

Приоритет логических операторов

- Оператор && имеет больший приоритет, чем ||
- Поэтому эквивалентно:
  a > 5 || c > 5 && d < 3
  a > 5 || (c > 5 && d < 3)
- Приоритет логических операторов, ниже, чем у операторов сравнения
- Поэтому сначала вычисляются подвыражения, например
  a > 5, c > 5, d < 3
- А уже для них производится комбинирование при помощи связок

Приоритет логических операторов

- Имеет смысл ставить скобки для повышения читаемости, но в меру:
- `a > 5 || c > 5 && d < 3` // плохо читаемо
- `(a > 5) || (c > 5 && d < 3)` // читается лучше
- `(a > 5) || ((c > 5) && (d < 3))` // чрезмерное // количество скобок

- Если у вас в одном условии есть && и ||, то обязательно ставьте скобки, чтобы был виден их взаимный приоритет

Свойства логических связок

- **Законы де Моргана:**

  !(a && b) ⇔ !a || !b

  !(a || b) ⇔ !a && !b

- **Дистрибутивность:**

  a && (b || c) ⇔ (a && b) || (a && c)

  a || (b && c) ⇔ (a || b) && (a || c)

Обращение операторов сравнения

- Если хотим получить отрицание операторов сравнения

Оператор	Обращение
a == b	a != b
a > b	a <= b
a < b	a >= b

- Строгое сравнение заменяется нестрогим

Задача

- Практика у доски – обращение сложного логического выражения (с логическими связками и арифметическими операциями)

Свойства логических связок

- **Идемпотентность:**

  A && A ⇔ A

  A || A ⇔ A

- **Двойное отрицание:**

  !!A ⇔ A

- **Аксиомы:**

  A && !A ⇔ false

  A || !A ⇔ true

Свойство && и ||

- Данные логические связки вычисляют результат слева направо, и прекращают вычисления, если результат уже определился
- Пример:
- 3 > 5 && a > b // будет false, второе условие
  // даже не будет проверяться
- 3 < 5 || a > 5 // будет true, второе условие
  // даже не будет проверяться
- За счет этого, эти связки некоммутативны,
  т.е. A && B не всегда ⇔ B && A

Свойство && и ||

- Свойство, что логические связки && и || могут не вычислять все операнды, часто используется на практике

- Тогда такое условие будет работать правильно:
- x != 0 && y / x > 0

- А такое упадет, если x будет равен 0:
- y / x > 0 && x != 0

Условный оператор if

- ```
  if (a > 5) {
      System.out.println("Внутри if");
  }
  ```

- Код внутри блока if выполняется только если условие в круглых скобках истинно (равно true)
- Т.е. когда сюда дошел код, то вычисляется выражение-условие. Если результат true, то выполняется код в фигурных скобках
  ```

Условный оператор if

- if (логическое выражение) инструкция
- Здесь инструкция – это либо 1 строка кода, либо целый блок кода внутри фигурных скобок
- Пример:
- ```
  if (a > 5) b = 3; // внутри if одна команда
  ```
- ```
  if (a > 5) {
      b = 3;
      c = 4;
  }
  ```
- Внутри блока делается отступ размером в 1 табуляцию

Условный оператор if

- Следует всегда использовать блок в фигурных скобках, даже для одной команды, чтобы не ошибиться

- ```
  if (a > 5)
    b = 3;   // выполняется если a > 5
    c = 4;   // выполняется ВСЕГДА - ошибка
  ```
  ```

Условный оператор if и ;

- Пример:
- ```
  if (a > 5) b = 3; // внутри if одна команда
  ```
- ```
  if (a > 5) {
      b = 3;
      c = 4;
  }
  ```
- Заметим, что после фигурных скобок if не ставится точка с запятой

Условный оператор if и ;

- Пример:
- ```
  if (a > 5) b = 3;   // внутри if одна команда
  ```
- Заметим, что после условия в скобках точка с запятой тоже не ставится
- Это очень важно, потому что отдельно стоящая точка с запятой считается пустой командой, которая ничего не делает
- ```
  if (a > 5) ; b = 3;
  ```
- В таком коде b = 3 выполнится всегда, а внутри if будет пустая инструкция

if-else

- if может содержать необязательную часть else (иначе) – какой код выполнить, если условие в if является ложным

- if (логическое выражение)
  инструкция1
  else
  инструкция2

Пример if-else

- ```
  if (a > 5) {
      System.out.println("a > 5");
  } else {
      System.out.println("a <= 5");
  }
  ```

- Условие вычисляется 1 раз, и если оно оказалось true, то мы заходим в ветку if, иначе – в ветку else
  ```

if-else против отдельных if'ов

- if-else намного лучше, чем 2 отдельных if'а
- ```
  if (a > 5) {
      System.out.println("a > 5");
  } else {
      System.out.println("a <= 5");
  }
  ```

Здесь всего 1 проверка условия – это быстрее и проще

- ```
  if (a > 5) {
      System.out.println("a > 5");
  }
  if (a <= 5) {
      System.out.println("a <= 5");
  }
  ```

Здесь всегда 2 проверки условий – это дольше и сложнее. Еще нужно не ошибиться во втором условии

if-else против отдельных if’ов

- Кроме того, в if-else есть гарантия, что код точно зайдет ровно в одну из веток
- Когда у вас 2 отдельных if’а, то гипотетически код может не зайти ни в один из них (если оба условия окажутся false)
- Либо гипотетически зайти в обе ветки (если оба условия будут true)

- Поэтому, учитывая предыдущие аргументы за if-else, всегда старайтесь использовать if-else, если он подходит

Вложение if’ов

- Ветвления (и другие конструкции, которые мы проходили и пройдем) можно вкладывать друг в друга любым образом
- ```
  if (a > 5) {
      int b = a + 5;

      if (b < 33) {
          System.out.println("1");
      } else {
          System.out.println("2");
      }
  }
  ```
  ```

Составление цепочек if-else

- Если нужно больше вариантов, чем 2, то можно составить цепочку if’ов
- ```
  if (a > 5) {
      System.out.println("a > 5");
  } else if (a == 5) {
      System.out.println("a = 5");
  } else {
      System.out.println("a < 5");
  }
  ```
- Здесь у нас 3 варианта, и код обязательно зайдет ровно в один из них
  ```

Составление цепочек if-else

- Веток else if может быть сколько угодно
- ```
  if (a > 5) {
      System.out.println("a > 5");
  } else if (a == 5) {
      System.out.println("a = 5");
  } else if (a == 4) {
      System.out.println("a = 4");
  } else {
      System.out.println("a < 4");
  }
  ```
- Ветка else, если она есть, должна быть последней
- Но ее может не быть, если мы не хотим ничего делать, если ни одно условие не подойдет
  ```

Составление цепочек if-else

- На самом деле цепочка if'ов не является отдельной конструкцией языка
- Это всего лишь вложенные if-else, в которых в else не написали фигурные скобки
-
```
if (a > 5) {
    System.out.println("a > 5");
} else {
    if (a == 5) {
        System.out.println("a = 5");
    } else {
        System.out.println("a < 5");
    }
}
```

Такой вариант иногда тоже подходит

Но вариант с цепочкой, где убраны выделенные фигурные скобки, и сделано форматирование кода, читается лучше

Цепочка if'ов против отдельных if'ов

- Цепочка if'ов лучше, чем отдельные if'ы по тем же причинам, что if-else лучше, чем 2 отдельных if'а
- Только здесь выгода еще больше
-
```
if (a > 5) {
    System.out.println("a > 5");
} else if (a == 5) {
    System.out.println("a = 5");
} else {
    System.out.println("a < 5");
}
```

В варианте с отдельными if всегда будет 3 проверки, а в варианте с цепочкой – одна или две

-
```
if (a > 5) {
    System.out.println("a > 5");
}
if (a == 5) {
    System.out.println("a = 5");
}
if (a < 5) {
    System.out.println("a < 5");
}
```

Цепочка if’ов против отдельных if’ов

- Кроме того, в цепочке каждый последующий if выполняется только если предыдущие if были false
- Поэтому часто бывает возможно упростить условия с учетом этого

Задача

- Прочитайте с консоли целое число
- Если оно положительное, то напечатайте в консоль строку – «Данное число - положительное»
- Если оно 0, то распечатайте, что оно 0
- Если отрицательное, то, распечатайте, что число отрицательное
- Используйте цепочку if’ов

Задача

- Прочитайте с консоли целое число
- Напечатайте, что число четное, если оно четное. И что нечетное, если оно нечетное
- И что кратно 5, если кратно 5. И что не кратно 5, если оно не кратно 5

Проверка boolean на true/false

- Часто бывает нужно проверить boolean переменную на равенство true или false
- Пусть у нас есть булева переменная **x**, она уже присвоена

- Конечно, можно проверять так:
- ```
  if (x == true) {
      // код
  }
  ```
- ```
  if (x == false) {
      // код
  }
  ```
- Но так делать не принято

Проверка boolean на true/false

- Нужно делать так:
- ```
  if (x) {
      // x равно true
  }
  ```
- ```
  if (!x) {
      // x равно false
  }
  ```

- Потому что это и так boolean – если **x** равен true, то условие if будет true
- Если нужна проверка на false, то просто применяем отрицание

Выражения

Выражения

- **Выражение** – это конструкция языка, которая выдает результат некоторого типа
- Являются выражениями:
  - Литералы
  - Переменные
  - Комбинации выражений через операторы (например, через арифметические операторы, операторы сравнения, логические операторы и др.)
  - Многие функции, например, Math.sqrt
- Не являются выражениями:
  - Некоторые функции, например, System.out.println
  - Некоторые конструкции языка, например, if

Выражения

- Выражения используются в программировании повсеместно
- Везде, где нам требуется какое-либо значение, мы можем указать выражение соответствующего типа (или типа, который неявно приводится к нужному типу)
- int x = 3 + 5;
- Например, здесь в правой части может быть любое выражение, результат которого является int’ом, либо неявно приводится к int

Выражения и функции

- Также все функции могут принимать выражения
- Например:
  - ```
    double radius = Math.sqrt(3);
    double area = Math.PI * Math.pow(radius, 2);
    ```
- Здесь функция Math.pow принимает два выражения – переменную **radius** и литерал **2**
- Но вместо них можно использовать любые выражения нужных типов:
  - ```
    double area = Math.PI * Math.pow(Math.sqrt(3), 2);
    ```
- Когда функция вызывается, то сначала вычисляются выражения-аргументы, а сама функция вызывается уже от этих результатов

Тернарный оператор

Тернарный оператор

- Является альтернативой для if-else
- **Запись**:
  - логическое выражение ? выражение1 : выражение 2
- Выражения 1 и 2 должны возвращать значения одного и того же типа (или совместимых типов)
- **Пример** - определение максимума из двух чисел:
- ```
  int max;
  if (x > y) {
      max = x;
  } else {
      max = y;
  }
  ```

```
int max = (x > y) ? x : y;
```
```

Отличия тернарного оператора от if

if	Тернарный оператор
Может не иметь else	Всегда 2 ветки
В каждой ветке может быть блок кода с любыми командами	В ветке может быть только по одному **выражению**
Не является **выражением**	Является **выражением**

- Как видно, тернарный оператор достаточно узкоспециализированный
- Поэтому он применяется не очень часто
- Но в простых случаях он подходит, и его можно использовать

Домашняя работа

Задача на дом «Max/min»

- Прочитать из консоли два целых числа
- Вывести наименьшее и наибольшее из них

- Сделать данную задачу при помощи if-else и при помощи тернарного оператора
- В версии с тернарным оператором допускается warning про использование тернарного оператора

Длина строки

- ```
  String string = "Hello";
  int stringLength = s.length();    // 5
  ```

- Length – с англ. длина
  ```

Проверка строк на равенство

- В Java для строк оператор == не подходит для сравнения строк
- Можете проверить следующий код:
-

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

String s = "Hello";

System.out.println("Введите строку:");
String userLine = scanner.nextLine();

if (userLine == s) {
    // false, даже если ввести Hello
}
```

Проверка строк на равенство

- Поэтому проверять строки на равенство нужно другим способом, при помощи команды **equals**:

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

  String s = "Hello";

  System.out.println("Введите строку:");
  String userLine = scanner.nextLine();

  if (s.equals(userLine)) {
      // true если ввели Hello
  }
  ```
  ```

Задача на дом «Пароль»

- В программе объявить строковую переменную, хранящую пароль
- С консоли прочитать строку, сравнить её с этим паролем. Если строка совпала (проверять при помощи **equals**), то выдать сообщение, что пароль верный
- Если строка не совпала с паролем, и её длина (использовать **length**) больше длины пароля, то сказать что пароль неверный и строка слишком длинная
- Если строка не совпала с паролем, и её длина меньше, то сказать, что пароль неверный строка слишком короткая
- Иначе сказать, что пароль неверный

Задача на дом «Високосный год»

- Прочитать с консоли год и вывести в консоль, является он високосным или нет

- Старайтесь использовать логические связки, если это возможно

Задача на курс «Площадь треугольника»

- Прочитать с консоли координаты трёх точек на плоскости:
  (x1, y1), (x2, y2), (x3, y3)
- Вычислить и вывести на экран площадь данного треугольника
- Для вычисления площади можно воспользоваться формулой Герона:
- $S_\Delta = \sqrt{p(p-a)(p-b)(p-c)}$,
  где p –полупериметр треугольника p=(a+b+c)/2,
  a, b, c –длины сторон треугольника
- Проверить на случай, когда эти точки лежат на одной прямой – в этом случае вычислять площадь не нужно, а нужно вывести сообщение об этом
- Для вычисления корня использовать команду
  Math.sqrt(значение)

Задача на курс «Возраст»

- Программа просит ввести пользователя свой возраст от 1 до 112 включительно, после чего выводит сообщение «Вам x лет»
- При этом учесть, что для разных чисел разные склонения
- Например, «3 года», «99 лет» и т.д.

- Если введут слишком малое или слишком большое число, то выведите, что «Вы слишком малы» или стары

- Старайтесь использовать логические связки, если это возможно

Задача на курс «Квадратное уравнение»

- Прочитать с консоли коэффициенты a, b и с квадратного уравнения $ax^2 + bx + c = 0$ и найти решение этого уравнения
- Не забыть рассмотреть все 3 случая – когда есть 2 корня, 1 корень и нет решений
- Программа должна работать даже если уравнение не квадратное (а равен 0), например, решать линейное уравнение и т.д.

Задача на курс «Следующая дата»

- Программа запрашивает сегодняшнюю дату и выдает дату следующего дня
- Например, входные данные: 31 12 2015, на выходе: 01.01.2016
- День, месяц и год можно запрашивать у пользователя с консоли по очереди
- Еще сделать проверку даты на корректность