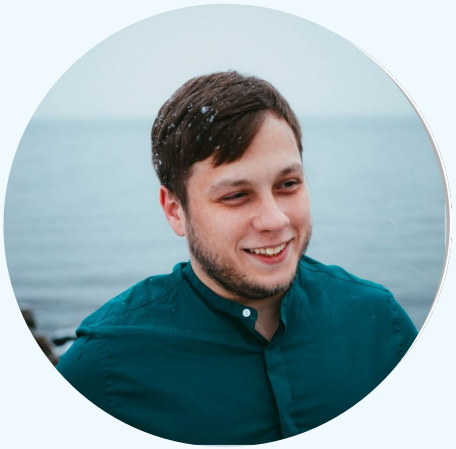
нетология

Условные операторы и циклы

Филипп
Воронов



Филипп Воронов

Teamlead, VK Group

@Филипп Воронов

План занятия

1. Условные операторы
2. Сравнение примитивных и ссылочных типов
3. Циклы
4. Прерывания циклов
5. Как считывать и выводить данные в цикле?

Условные операторы

Условные операторы

Давайте рассмотрим шаги выполнения программы:

1. Прочитать первое значение из консоли;
2. Выполнить операции над ним;
3. Прочитать второе значение из консоли;
4. Выполнить операции над первым и вторым значениями;
5. Вывести результат на экран.

Эти шаги последовательны и исключают вариативность.

Условные операторы

А что если мне нужно выполнять разные действия после первого шага, в зависимости от прочитанного на первом шаге значения:

1. Прочитайте первое значение из консоли;
2. Проверьте условия:
    - если на 1-м шаге введен ноль, завершите программу;
    - если на 1-м шаге введен не ноль, продолжите выполнение;
3. Прочитайте еще одно значение;
4. Выполните операции над первым и вторым значениями;
5. Выведите результат на экран.

Условные операторы

Проблему ветвления или вариативности решают условные операторы:

```
1. if
2. switch
3. тернарный оператор
```

Оператор `if`

`if` — самый часто используемый условный оператор. Он проверяет условие, указанное в качестве аргумента `if (условие)`.

Если результат вычисления условия верен, то программа продолжит выполнение кода, написанного внутри блока `{ код }`. Если не верен, то пропустит блок.

Пример: Если температура чайника достигла 100 градусов выполни код (выключи его):

```
if (temperatureTeapot == 100) {
    // Код сработает, только если переменная temperatureTeapot = 100;
}
```

Оператор `if`

Если нужно проверить несколько условий, к оператору `if` добавляется оператор else (иначе). Все блоки кода размещаются внутри `{ код }`.

Пример: Если температура чайника достигла 100 градусов, выполни код (выключи его). Если нет, то продолжай кипятить воду в нем.

```
if (temperatureTeapot == 100) {
    // Код в этом блоке сработает, только если переменная temperatureTeapot = 100;
} else {
    // Код в этом блоке сработает, только если переменная temperatureTeapot не
равна 1;
}
```

Оператор `if`

Для проверки более чем 1 условия операторы `if else` размещаются друг за другом.

Пример: Если температура чайника достигла 100 градусов, выполните код (выключи его). Если она равна 85 градусов, выведите на дисплей осталось 20 секунд. Если она равна 60 градусов, включите дополнительный тен. Если же ни одно условие не подошло, то просто продолжайте кипятить воду в нем.

```
if (temperatureTeapot == 100) {
    // Код в этом блоке сработает, только если переменная temperatureTeapot = 100;
} else if (temperatureTeapot == 85) {
    // Код в этом блоке сработает, только если переменная temperatureTeapot = 85;
} else if (temperatureTeapot == 60) {
    // Код в этом блоке сработает, только если переменная temperatureTeapot = 60;
} else {
    // Код в этом блоке сработает, если ни одно условие выше не оказалось верным;
}
```

Оператор `if`

В некоторых случаях можно использовать короткую запись оператора `if` без `{ код }`, но только для однострочных инструкций.

Пример:

```
if (temperatureTeapot == 0) System.out.println("Температура чайника 0 градусов");
```

или более сложный пример:

```
if (temperatureTeapot == 100)
    System.out.println("Чайник вскипел");
else
    System.out.println("Температура чайника еще меньше 100 градусов");
```

Оператор `switch`

`switch` – еще один условный оператор, в качестве аргумента принимает переменную и сравнивает ее значение в инструкциях оператора `case`.

Дополнительно оператор `switch` имеет инструкцию `default`, определяющую вариант по умолчанию.

Аналогично `else` срабатывает в случае, если ни один из вариантов оператора `case` не совпал.

Визуально структура оператора `switch` отличается от оператора `if`, но эти операторы взаимозаменяемы, и в конечном счете байт-код после компиляции будет одинаков.

Оператор `switch`

Пример: В зависимости от температуры холодильника включите или отключите компрессор охлаждения в нем.

```
switch (refrigeratorTemperature) {
    case 0:
        // Код в этом блоке сработает, только если переменная
refrigeratorTemperature = 0;
        break;
    case -20:
        // Код в этом блоке сработает, только если переменная
refrigeratorTemperature = -20
        break;
    default:
        // Код в этом блоке сработает, если ни одно условие выше не оказалось
верным;
}
```

Оператор `switch`

В коде оператора `switch` мы объявили еще незнакомый оператор `break`. Когда выполнение программы доходит до него, интерпретатор Java выходит из блока `case`.

Если не написать `break` после выполнения кода в блоке `case`, мы автоматически перейдем к блоку кода следующего `case`. При этом проверяться условие уже не будет.

Поэтому важно после каждого case не забывать добавлять оператор `break`.

Тернарный оператор

Тернарный оператор — условный операторa без тела кода. Он используется, когда нужно присвоить значение переменной или вернуть значение из метода, не создавая избыточный код.

Конструкция состоит из условия `( условие )` и двух выражений. Первое выражение сработает, если условие верно, а второе — если ложно.

Выражения и условие разделены знаком вопроса `?`.

Тернарный оператор

Пример: Если датчик температуры холодильника возвращает значение больше или равно нулю, подайте питание на компрессор охлаждения.

Если меньше нуля — не подавайте питание.

```
int switchOnRefrigiratorEngine = (refrigeratorTemperature >= 0) ? 1 : 0;
```

Если `(refrigeratorTemperature >= 0)` верно, то новое значение переменной `switchOnRefrigiratorEngine` будет равно `1`.

Если ложно, то новое значение переменной `switchOnRefrigiratorEngine` будет равно нулю.

То есть без явного указания `if` можно создать условие присвоения значения.

Как правильно сравнивать переменные

В Java все типы данных делятся на две группы:

- Примитивные типы;
- Ссылочные типы.

Все примитивные типы сравниваются с помощью оператора `==`, а у всех ссылочных типов есть метод для их сравнения: метод `equals`.

Такая необходимость возникла вследствие того, что ссылочные типы более сложные, и их область хранения внутри JVM отличается от расположения примитивных типов.

К примитивным типам относятся почти все числовые типы например `byte` и `int`.

Как правильно сравнивать переменные

Пример: Сравним температуру в микроволновой печи с температурой еды в ней.

```
int microvaweTemperature = 80;
int foodTemperature = 10;
 if (microvaweTemperature == foodTemperature) {
     System.out.println("Еда согрета");
 }
```

Строковый тип `String` является ссылочным, поэтому для его сравнения нужно использовать метод `equals`.

Пример: Сравним метод разогрева еды, выбранный пользователем, и тем, что предоставлен на выбор.

```
String userChoise = "экологично";
 if ("быстро".equals(userChoise)) {
     System.out.println("Выбран быстрый режим");
 } else if ("экологично".equals(userChoise)) {
     System.out.println("Выбран eco режим");
   }
```

Циклы

Циклы

В Java, как и во многих других языках, для выполнения одного и того же кода много раз, помимо методов есть циклы.

Цикл — это повтор выполняемого кода.

Циклы бывают нескольких видов:

1. `while`;
2. `do while`;
3. `for`;
4. `foreach`.

Цикл `while`, `do while`

Цикл `while` повторяет оператор или группу операторов, пока заданное условие является верным.

Цикл проверяет условие до выполнения тела цикла, поэтому цикл `while` называется циклом с предусловием.

Пример: Выключите чайник, когда вода достигнет температуру в 100 градусов по цельсию.

Если нет, продолжите подавать питание и кипятить воду.

```
int teapotTemperature = 27;
while (teapotTemperature < 100) {
    heatTeapot();
    teapotTemperature = getTeapotTemperature();
    System.out.println("Текущая температура чайника: " + teapotTemperature);
}
```

Пока температура чайника не достигнет 100 градусов, цикл будет выполняться.

Цикл `while`, `do while`

С помощью цикла `do while` выполняется цикл `while`, за исключением того, что он проверяет условия в конце тела цикла. Поэтому цикл называется циклом с постусловием.

Пример: Теперь мы хотим сварить макароны и будем их класть в кастрюлю до тех пор, пока она не заполнится доверху.

```
int pasta = 0; //Количество макарон в кастрюле
int potSize = 1000; //Емкость кастрюли;
do {
    pasta = pasta + 100;
    System.out.println("Количество пасты в кастрюле: " + pasta);
} while (pasta < potSize);
```

Переменная `pasta` будет увеличиваться до тех пор, пока ее значение не сравняется со значением переменной `potSize`.

Цикл `do while` используется в случаях, если операторы в цикле нужно выполнить минимум 1 раз.

Цикл `for`

Еще одним оператором цикла является оператор `for`, он выполняет последовательность кода в нем столько раз, сколько определено в его переменной счетчика и называется циклом со счетчиком.Такой цикл хорошо подходит, когда мы точно знаем, сколько раз мы хотим выполнить наш код.

Пример: Если мы хотим порезать картошку на 1000 маленьких кусочков, мы будем использовать цикл `for`.

```
for (int i = 0; i < 1000; i++) {
    System.out.println("Количество кусочков картошки: " + i);
}
```

`i` — переменная счетчика, `i++` — операция пост инкремент, увеличение переменной на `1`.

Плюс такого объявление цикла в том, что не нужно заранее в коде добавлять дополнительные переменные счетчика: как `i` из примера, они описываются и доступны только в пределах цикла `for`.

Цикл `foreach`

Одной разновидностью цикла `for` является цикл `foreach`. Он используется для перебора элементов массива или коллекции. Его сигнатура не включает присваивания начального значения и ее последующего инкремента.

Пример: Для того чтобы вывести слово Борщ, перебрав все элементы массива, достаточно использовать цикл с перебором элементов этого массива.

Разница между `for` и `foreach`

Напишем код с использованием цикла `for`:

```
char[] borsh = {'Б','о', 'р', 'щ'};
for(int i = 0; i < borsh.length; i++) {
    System.out.print(borsh[i]);
}
```

А теперь напишем с использованием `foreach`:

```
char[] borsh = {'Б','о', 'р', 'щ'};
for(char letter : borsh) {
    System.out.print(letter);
}
```

В нашем примере цикл `foreach` не требует дополнительных переменных, отвязанных от логики выражения, но при этом улучшает читаемость кода.

Чаще всего цикл `for` применяется для перебора элементов структур данных.

Прерывания циклов

Иногда встречаются ситуации, когда нужно срочно прервать выполнение цикла или пропустить итерацию, для этого были введены два оператора: `continue` и `break`.

Первый оператор `continue` пропускает выполнение итерации в цикле.

Пример: Мы хотим приготовить суп, и будем выбирать каждую вторую (четную) морковь из ведра.

```
for (int i = 0; i < 10; i++) {
    if (i % 2 != 0)
        continue;
    System.out.println("Морковь номер: " + i);
}
```

При каждой проверке `if` мы проверяем, делится ли наше число без остатка на `2`. Если делится, значит `i` четное число.

Прерывания циклов

Пример: При варке нашего супа мы добавляли разные овощи. При добавлении лука мы решили, что достаточно овощей.

```
String [] vegetables = {"курица","петрушка","лук","сельдерей"}
for (String vegetable : vegetables) {
    if ("лук".equals(vegetable))
        break;
    System.out.println(vegetable);
}
System.out.println("Время варить!");
```

Этот цикл будет выполняться, пока значение `vegetable` не будет равно значению `"лук"`.

Как только выполнится условие оператора `if`, сработает вызов оператора `break`, программа выйдет из цикла и напечатает `"Время варить!"`.

Как считывать и выводить данные в цикле?

Для того чтобы выводить и считывать данные в бесконечном цикле, нужно написать следующий код:

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

while (true) {
    System.out.println("Введите значение"); //Выводим значение в консоль
    String value = scanner.nextLine(); //Считываем значение из консоли в переменную
    //TODO
}
```

Как считывать и выводить данные в цикле?

Чтобы выйти из такого цикла нужно задать условие выхода, например, ввод слова `end`:

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

while (true) {
    System.out.println("Введите значение"); //Выводим значение в консоль
    String value = scanner.nextLine(); //Считываем значение из консоли в
переменную
    //TODO
    if ("end".equals(value)) { //если введенное значение равно `end` выходим из
цикла
        break;
    }
}
```

Как считывать и выводить данные в цикле?

Для того чтобы выводить и считывать данные в цикле со счетчиком, напишем следующий код:

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

for (int i = 0; i < 10; i++) {
    System.out.println("Введите значение"); //Выводим значение в консоль
    String value = scanner.nextLine(); //Считываем значение из консоли в
переменную
}
```

В этом случае нам не нужно условие выхода. Через 10 итераций цикл завершится, и программа выйдет из него.

Чему мы научились

- Рассмотрели условные операторы;
- Изучили циклы;
- Узнали, как пропустить выполнение итерации;
- Узнали, как прервать выполнение работы циклом;
- Узнали, как правильно сравнивать типы;
- Узнали, как читать данные в цикле.

Домашнее задание

Давайте посмотрим ваше домашнее задание.

- Вопросы по домашней работе задавайте **в чате** мессенджера Slack.
- Задачи можно сдавать **по частям**.
- Зачёт по домашней работе проставляется после того, как **приняты все задачи**.

нетология

Задавайте вопросы и пишите отзыв о лекции!

Филипп Воронов

@Филипп Воронов