Данные

Надеюсь вы помните, как мы простыми словами описали программу для робота, в которой у нас было упомянуто очень важное, но сразу не очень заметное понятие. Мы передавали нашему роботу ДАННЫЕ. В нашем случае это было число метров, на которое робот должен проехать вперед и количество градусов, на которое он должен был повернуть.
В реальных программах мы постоянно работаем с данными. Мы их получаем внутри самой программы путем каких-либо расчетов, получаем данные от пользователя, позволяя ему вводить данные через какие-то поля, получаем данные из внешних источников. Если попробовать систематизировать источники для данных, то их можно разделить на следующие категории:

1. Данные внутри программы

С одной стороны здесь все очевидно — программа должна работать с данными. И внутри программы данные должны быть. С другой стороны — все очень непросто. Данные сами по себе могут иметь достаточно сложную структуру. Например, мы хотим написать программу для расчета движения тела с заданной скоростью из какой-то точки на плоскости в другую. В данном случае нам будет достаточно ввести координаты X и Y, время T и скорость V. Далее мы сможем использовать несложные формулы для расчетов и хранения наших данных для последующего вывода (формулы предлагается взять из учебника физики или математики). Но если тело будет не одно, а 5 или 10 ? Или в общем случае нам заранее

неизвестно сколько тел мы будет обсчитывать. Тогда нам потребуется более сложная структура данных. Было бы здорово, чтобы для описания состояния одного тела мы могли вы выделить какую-то группу данных, а потом объединили их в еще какую-то структуру для хранения. Например структуру BODY содержала бы нужные нам X,Y,V, а структура ALL_BODY содержала бы список структур BODY. Как видим, само построение структур данных задача не всегда тривиальная. Очень интересная книга по данным и алгоритмам их обработки была написана Никлаусом Виртом — «Алгоритмы и структуры данных». Можете поискать в интернете. Но скорее всего прочитать сразу все вы не сможете — книга несколько академична. Но тем не менее ее можно и нужно посмотреть.

Важно понять один момент — ДАННЫЕ НЕ ОГРАНИЧИВАЮТСЯ ПРОСТЫМИ ВЕЩЕСТВЕННЫМИ ЧИСЛАМИ. Например, данные о гражданине может включать в себя достаточно много полей — имя, фамилия, отчество, дата рождения, паспортные данные и многое другое. И эти данные надо уметь организовывать в удобные для обработки структуры — по таким структурам удобно осуществлять поиск, их удобно отображать, хранить. С их помощью удобно добавлять, редактировать, удалять, работать с целыми группами.

Организация данных весьма важный вопрос. На мой взгляд, именно при решении задач организации данных появилось объектно-ориентированное программирование, о котором пойдет речь при изучении Java.

2. Внешние данные

Внешние данные — это данные, которые находятся где-то вне программы. С помощью уже готовых программ (подпрограмм, пакетов, технологий) эти данные можно получить внутрь нашей программы. Стоит обратить внимание, что внешние данные все равно придется для начала разместить во внутренних данных. И только потом ими можно будет пользоваться. Давайте более подробно остановимся на том, откуда же можно брать данные. Ответ на этот вопрос дает нам список тех технологий (пакетов, подпрограмм), которые используются в современных языках программирования (в том числе и в Java). Итак:

2.1 Файлы

Достаточно очевидное хранилище данных. Наверняка многие из вас создавали документы в программе «Блокнот», MS Word, Excel и многих других. Каждый документ сохраняется в файле. Это и есть наши внешние данные. Совсем необязательно, что ваши файлы будут содержать столь сложные структуры, как документ в формате Word, но тем не менее нахождение данных в файле — важный момент.

2.2 Внешние программы

Внешние программы — это не только программы, которые запущены на Вашем компьютере — это программы, которые запущены на других компьютерах (хотя может быть и на том же самом — но это ДРУГИЕ программы). В эру сети Интернет взаимодействие программ становится очень важным элементом и возможности взаимодействия программ необходимо изучать. Разумеется в Java и в других современных языках есть целый ряд технологий, позволяющих осуществлять такое взаимодействие.

2.3 Базы данных

Если быть точным, то работа с базами данных может быть включена как в раздел «Файлы», так и в раздел «Внешние программы». Базы данных могут быть организованы в виде простых файлов или в виде специальной программы, которая обрабатывает внешние запросы от других программ (такая организация сегодня является наиболее предпочтительной даже в случае, если база данных используется локально).

Переменные и типы данных в Java

Зачем нужны переменные ?

На самом деле это достаточно важный вопрос. Большинство современных программистов не очень задумываются, насколько важным является введение переменных. Когда алгоритмических языков не было, программистам надо было думать не только над тем, как обрабатывать данные — надо было думать над тем, где эти данные хранить. Перед программистом был просто кусок памяти, внутри которого можно и нужно было разместить команды и данные. Каждое число, символ или строку надо было разместить в ячейках памяти по определенному адресу. Что было кропотливой и достаточно скучной работой.
Мало того — надо было еще помнить, какого типа данные лежат в той или иной ячейке. Потому как хранить целое или вещественное число надо было по-разному. И обрабатывать тоже по-разному. Чтобы не делать эту малоинтересную работу постоянно, в алгоритмических языках появилось такое понятие — переменная. При компиляции программ каждая переменная (со своим именем) размещается в определенных ячейках и ее обработка зависит от ее типа. Причем соблюдение этих правил берет на себя программа/компилятор, а программист может сосредоточиться на более интересных вещах.
Это, на мой взгляд, было здорово — программист теперь не думал, где и как хранить данные. Он просто ОБЪЯВЛЯЛ переменную нужного типа. И в дальнейшем в программе обращался к ней по имени. А выделение памяти под нее, правильность ее обработки брал на себя компилятор.
И теперь программисту надо было сделать только одно — объявить переменную с нужным типом. На самом деле есть языки, где этого делать не надо, но на мой взгляд, это хорошо работает в небольших программах. Сам по себе очень неоднозначный вопрос и много копий было сломано и немало «священных войн» (holy war — холивар) можно найти на просторах Интернет по этому поводу.

В Java объявлять переменные надо обязательно. В самом простом виде это выглядит во так:

<тип> имя;

Имя переменной в Java включает следующие символы — латинские буквы (большие и маленькие), цифры, знак доллара «$» и знак подчеркивания «_». Имя не должно начинаться с цифры. Длина имени имеет ограничения, но такую длины вы скорее всего использовать не будете — речь идет об очень больших величинах.

Таким образом корректными будут следующие имена:

```
counter
summa
```

```
q123$t_A
_$123
_anton12
```

Данные в Java условно можно разделить на два больших раздела:

1. Элементарные типы данных
2. Сложные типы данных — классы

Элементарные типы данных

Элементарными типами являются следующие:

- целые числа
- вещественные числа
- единичные символы
- булевы или логические типы

Целые числа — это обычные целые числа. Но внутри Java их несколько.

- byte — число от -128 до 127 (длина 1 байт)
- short — число от -32,768 до 32,767 (длина 2 байта)
- int — число от -2,147,483,648 до 2,147,483,647 (длина 4 байта)
- long — число от -9,223,372,036,854,775,808 до +9,223,372,036,854,775,807 (длина 8 байт)

Вещественные числа — это числа с десятичной дробью.

- float — число от 1.40129846432481707e-45 до 3.40282346638528860e+38 (длина 4 байта)
- double — число от 4.94065645841246544e-324d до 1.79769313486231570e+308d (длина 8 байт)

Вещественное число можно записывать в форме с десятичной точкой. Например так:

```
10.5
```

```
967.956
```

Также число может быть записано со степенью десяти. Например 1.4 на 10 в степени 5 можно записать так:

```
1.4e5
```

Или так

```
1.4E5
```

Как видите, можно писать «e» или «E». Число перед «E» называется мантисса, после — показатель степени числа 10
Здесь надо учесть, что из-за ограничений по длине при сложении чисел с большой положительной степенью (например +40) и очень большой отрицательно (например -60) маленькое число будет просто утеряно — на него просто не хватит разрядов в мантиссе (это 1.40129..). Т.е. количество чисел после запятой ограничено и младшие разряды просто не учитываются. Для учета таких ситуаций были разрабтаны специальные программы для суммирования массивов чисел — сначала складывались самые маленькие, чтобы накопить хоть какое-то значимое число, потом прибавлялись все больше и больше.

Символ
char — символ (длина 2 байта — для возможности использовать Unicode)
Символ записывается в одиночных кавычках. Например

```
'A'
'$'
'n'
```

Также символ может быть записан в цифровой форме. Здесь надо учесть, что для символа используется два байта. Например, символ ‘A’ можно записать в виде ‘\ u0041’. Символ ‘1’ — ‘\u0031’. Но число записывается не в десятичном, а в шестнадцатиричном коде. Таким образом можно использовать непечатные символы.

Логическое значение
boolean — может принимать всего два значение — true или false

Т.е. если я хочу объявить переменную типа int с именем counter, то это будет выглядеть так:

int counter;

Вещественное (double) с именем bigCounter так:

double bigCounter;

ВАЖНОЕ ЗАМЕЧАНИЕ:
Размер данных строго фиксирован — для любой реализации Java (под любую операционную систему) размер элементарных переменных всегда один и тот же. Что отличает Java от того же Си — там не все типы имеют точный размер.

Операторы

Сами по себе данные важны, но нам нужны инструменты для операции над ними. Тип данных определяет какие операции мы можем использовать. Давайте рассмотрим их по порядку:

Оператор присваивания

Наверно это самый популярный и важный оператор. С его помощью можно в переменную поместить нужное значение. Записывается он в виде знака «=». Давайте сразу напишем несложную программу — файл **Second.java**.

```
1 public class Second
2 {
3     public static void main(String[] arg) {
4         int counter;
5         counter = 99;
6         System.out.println(counter);
7     }
8 }
```

И теперь разберем те строки, которые находятся под строкой

```
1 public static void main(String[] arg) {
```

Я не хочу сейчас долго и муторно вам объяснять то, что находится выше — давайте пока ограничимся тем, что наша программа начинается так, как она начинается. У нее есть верхняя часть

```
public class Second
{
    public static void main(String[] arg) {
```

Пока единственное, на что следует обратить внимание — это то, что слово, которое стоит после **public class** должно быть такое же, как и имя у файла (но без расширения .java). В нашем случае это **Second**. Придет время и мы постараемся разобраться со всеми этими словами. Но потихонечку.

Также есть часть, которую мы как раз и будем рассматривать — внутри фигурных скобок после строки **public static void main(String[] arg) {**. В ближайшее время все наше внимание будет именно здесь.

И заключительная часть (две закрывающие фигурные скобки)

```
    }
}
```

Ну что же, поехали. Первая строка — объявляем переменную целого типа с именем «counter».

int counter;

Вторая строка — присваивание переменной counter значения 99.

counter = 99;

И наконец третья строка выводит значение переменной counter на экран

System.out.println(counter);

Думаю, что самое время обратить ваше внимание на еще один момент — практически каждая команда на языке Java обычно заканчивается знаком «;» (точка с запятой). Этот символ используется в подавляющем количестве языков как указание на окончание команды. Дальше мы будем так часто использовать этот символ, что вы станете его использовать автоматически. (Это будет несомненно полезным навыком).
Мало того — вы можете использовать символ ; без оператора вообще. Это так называемый «пустой оператор». Например вот так (обратите внимание, что файл теперь должен называться SecondEmpty.java)

```
```

```
public class SecondEmpty
{
    public static void main(String[] arg) {
        int counter;
        ;
        ;
        ; ;
        counter = 99;
        System.out.println(counter);
    }
}
```

И это совершенно корректная программа будет выполняться. Несмотря на то, что у нас несколько пустых операторов. Есть еще один важный момент — в команде присваивания вы можете использовать пробелы. Столько, сколько захотите. Вы можете даже переносить элементы этой команды на разные строки. Даже так:

```
counter
=
99
;
```

Но не увлекайтесь — иначе вашу программу будет слишком сложно читать.
Оператор присваивания можно использовать сразу при объявлении переменной. Наша программа в этом случае будет выглядеть вот так:

```
public class Second
{
    public static void main(String[] arg) {
        int counter = 99;
        System.out.println(counter);
    }
}
```

Предлагаю вам самостоятельно сделать полный цикл — редактирование, компилирование, запуск. Закрепите недавно приобретенные знания.

Арифметические операторы

Арифметические операторы, как вы уже наверняка догадываетесь, предназначены в основном для чисел. К этим операциям относятся достаточно понятные:

1. Сложение — «+»
Например, сложить два числа и присвоить результат третьему.

```
int a;
a = 23 + 87;
```

2. Вычитание — «-«
Например:

```
int b;
b = 99 - 44;
```

3. Умножение — «*»
Например:

```
int a = 12;
int b = 34;
int c = a * b;
```

4. Деление — «/»
Например:

```
int a = 12;
int b = 36;
int c = b / a;
```

С делением все не так просто, как может показаться на первый взгляд. Когда вы используете вещественные числа, вы честно получите десятичное число с дробной частью. Но вот когда вы используете целые числа, то результат может вас несколько удивить. Попробуйте выполнить следующую программу (вы уже догадались, что файл должен называться Third.java):

```
public class Third
{
    public static void main(String[] arg) {
        int a = 12;
        int b = 34;
        int c = b / a;
        System.out.println(c);
    }
}
```

Результат будет : 2. На самом деле все логично — 12 помещается в 34 два раза. И остаток от деления получится 10.
Если погрузиться чуть глубже, то дело заключается в следующем — компилятор приводит все выражение справа от оператора присваивания к самому «вместительному» типу. Т.е. если бы мы делили double 34 на int 12, то получили бы 2.833333333.

```
public class Third
{
    public static void main(String[] arg) {
        int a = 12;
        double b = 34;
        double c = b / a;
        System.out.println(c);
    }
}
```

Обратите внимание на то, что нам придется и переменную «c» тоже объявить как double. Это необходимо, чтобы компилятор мог поместить «самый вместительный» тип (а он у нас double) в переменную для хранения результата.

5. Остаток от деления — «%»
Остаток от деления имеет смысл при делении целых чисел. Например, программа

```
public class Third
{
    public static void main(String[] arg) {
        int a = 12;
        int b = 34;
        int c = b % a;
        System.out.println(c);
    }
}
```

выводит результат 10.
Конечно же в одной команде можно использовать комбинацию разных операторов. В этом случае приоритет сложения, вычитания, умножения и деления определяется по стандартным правилам арифметики — произведение и деление имеют приоритет. Для изменения порядка вычисления используется такой же способ, как и в обычной математике — скобки. В Java используются круглые скобки (как и в остальных языках — я других скобок не встречал). Например:

```
1 public class Fourth
2 {
3     public static void main(String[] arg) {
4         double a = 12;
5         double b = 34;
6         double c = (b + a) / (a * b + 10);
7         System.out.println(c);
8     }
9 }
```

Вы догадались, что файл должен называться Fourth.java ?

Изменение значения переменной

Часто в программировании встречается ситуация, когда в переменную надо поместить значение, которое зависит от предыдущего значения этой же переменной. Например, надо увеличить значение переменной на 10.

```
1 int x = 20;      // Объявили переменную и присвоили ей значение 20
2 x = x + 20;      // Добавили к значению переменной еще 20
```

Конструкции, подобные этой настолько часто встречаются, что в Java были введены специальные операторы.

```
1 x += 20;     // Это эквивалентно  x = x + 20
2 x -= 15;     // Эквивалентно x = x - 15
3 x *= 2;      // x = x*2;
```

Более того — в программировании прибавление 1 к значению переменной настолько часто, что для этого было придуман специальный оператор.

```
x++;
++x;
```

Оба оператора соответствуют конструкции

```
x = x +1;
```

Но между первым и вторым существует одна разница, которую я продемонстрирую на примере. Обратите внимание на название AddOneFirst и AddOneSecond — именно так должны называться ваши файлы с текстом программ.

```
public class AddOneFirst
{
    public static void main(String[] arg) {
        int x = 10;
        int a = x++;
        System.out.println("x=" + x);
        System.out.println("a=" + a);
    }
}
```

Результат

```
x=11
a=10
```

```
public class AddOneSecond
{
    public static void main(String[] arg) {
        int x = 10;
        int a = ++x;
        System.out.println("x=" + x);
```

```
        System.out.println("a=" + a);
    }
}
```

Результат

```
x=11
a=11;
```

Как видим место расположения оператора ++ влияет на результат. В первом случае мы СНАЧАЛА ПРИСВОИЛИ переменной «a» значение из переменной «x» и только потом увеличили «x».
Во втором случае мы СНАЧАЛА УВЕЛИЧИЛИ переменную «x» а потом присвоили ее значение переменной «a».
Если вы задумаетесь о сдаче сертификата — там много коварных вопросов связанных с этим оператором. Попробуйте угадать результат следующих операторов:

```
int b = 5;
int a = 12;
int c = ++a - b++;
System.out.println(c);
int d = b++ * 2;
System.out.println(d);
```

Операция сложения для символов

Символьные данные тоже можно складывать через операцию сложения. Только надо учесть, что складываются символы по их цифровому значению. И получаем по сути целое число, которое можно преобразовать в символ.

Логические операции

Тип boolean на самом деле достаточно удобный для решения многих задач. Для него определены следующие операции:

1. Логическое сложение — «||»
В сложении если хоть одно слагаемое равно true, то результат тоже true. Например:

```java
boolean f1 = true;
boolean f2 = false;
boolean f3 = f1 || f2;
```

В переменной f3 результат равен true.

2. Логическое умножение — «&&»
В умножении если хоть одно слагаемое равно false, то результат тоже false. Например:

```java
boolean f1 = true;
boolean f2 = false;
boolean f3 = f1 && f2;
```

В переменной f3 результат равен false.

Для композиции нескольких выражений можно тоже использовать скобки. Попробуйте угадать результат следующей программы

```java
public class Fifth
{
    public static void main(String[] arg) {
        boolean a = true;
        boolean b = false;
        boolean c = true;
        boolean result = a && (b || c);
        System.out.println(result);
    }
}
```

3. Операция отрицания — «!»
Например:

```java
boolean a = true;
boolean b = !a;
```

Переменная b получит значение false.

Укороченное вычисление логических операций

Если вы создаете сложное логическое выражение, то нередко несмотря на его сложность, результат может быть известен после вычисления не всего выражения, а только его части. Например:

```
boolean a = true;
boolean b = false;
boolean c = true;

boolean d = a || (b && c);
```

Здесь можно видеть, что для вычисления переменной «d» совсем необязательно вычислять все — операция || указывает, что в данном случае достаточно того, что переменная «a» равна «true». Все остальное можно не вычислять. Т.к. первая часть в операторе ИЛИ дает нам TRUE, то зачем проверять (вычислять) вторую часть ? Там может быть что угодно, но результат все равно уже гарантировано будет TRUE. Это и есть принцип укороченного вычисления. Но иногда такое вычисление необходимо. Давайте снова рассмотрим пример. (Я очень хочу, чтобы вы читали код и понимали его — это очень полезный навык).
Так вот попробуйте угадать, какое значение переменной «b» будет выведено ?

```
public class ShortBool
{
    public static void main(String[] arg) {
        boolean a = true;
        int b = 0;

        boolean result = a || (++b > 0);
        System.out.println("B=" + b);
    }
}
```

Если вы подумали, что 0 — вы правильно подумали. Именно так — т.к. переменная «a» определяет заранее результат, то часть (++b > 0) даже не проверяется. Существует возможность заставить все-таки вычислять и вторую часть. Для этого надо использовать вместо «||» выражение «|».
Смотрим пример:

```
public class LongBool
```

```
{
    public static void main(String[] arg) {
        boolean a = true;
        int b = 0;

        boolean result = a | (++b > 0);
        System.out.println("B=" + b);
    }
}
```

Здесь выражение для вычисления переменной «result» выглядит немного иначе. И результат уже будет … 1. Точно такое же выражение можно использовать в случае «&&».
Насколько этот вариант вам будет подходить — смотрите сами. Лично я стараюсь не использовать такие конструкции — они могут запутать. Но иногда это удобно.

Комментарии

В любой программе есть код, который достаточно сложно понять. Для того, чтобы можно было вставить пояснения используются комментарии. В Java комментарии можно записывать двумя способами.
1. С помощью конструкции в виде двух символов «/*» для начала комментария и «*/» для окончания. Такая конструкция позволяет писать многострочные комментарии. Например:

```
/*
Пример для иллюстрации логических (булевых)
операций
*/
public class Fifth
{
    public static void main(String[] arg) {
        boolean a = true;
        boolean b = false;
        boolean c = true;
        boolean result = a && (b || c);
        System.out.println(result);
    }
}
```

2. Однострочный комментарий
Однострочные комментарии начинаются со строки «//» и длятся до конца строки.

```
public class Fifth
{
    public static void main(String[] arg) {
        boolean a = true;     // Объявление переменной a
        boolean b = false;
        boolean c = true;
        boolean result = a && (b || c);
        // Вывод результата на печать
        System.out.println(result);
    }
}
```

Отмечу один момент — пишите осмысленные комментарии. Не пишите после строки a = 10; комментарий типа «в переменную a помещаем число 10». Совершенно бесмысленна трата времени и пространства — это прекрасно видно из кода.

Мы познакомились с элементарными типами данных, посмотрели какие операции над ними можно производить. Знакомство с классами мы отложим на некторое время. А сейчас самое время познакомится с конструкциями для управления порядком выполнения операций. Управление порядком выполнения операций

28 comments to *Данные*

- Январь 12, 2015 at 19:12
  Эрик says:

  Здравствуйте!
  Подскажите, а зачем переменной b нужно быть тоже double?
  Разве не достаточно только только «c» быть double?
  b же у нас 34.
  Я новичок, поэтому для меня это непонятно:)

Reply

Январь 13, 2015 at 14:12
admin says:

В случае если b будет int, то сначала будет получен результат деления, у которого тип будет int — т.е. результатом будет 2. И уже потом он будет приведен к типу double. В итоге получим 2.0.

Reply

Июль 27, 2015 at 22:11
Art says:

Подскажите, почему сначала будет деление, если b будет int, правило скобок не работает?

Reply

Июль 28, 2015 at 06:04
admin says:

Добрый день. Хотелось бы подробнее — какой конкретно пример вызывает сложности ?

Reply

Июль 28, 2015 at 10:27
Art says:

Добрый день! public class Fourth:
Почему «a» и «b» переменные double, а не int?

Июль 28, 2015 at 11:18
admin says:

Потому что при делении целых чисел будет получатся целое число — например 13/5 даст результат 2, а не 2.6
Компилятор не будет создавать double, если нет ни одного числа double — все int и компилятор делает результат тоже int.

Июль 6, 2015 at 05:39
Grif says:

«Для учета таких ситуаций были разрабтаны специальные программы для сммирования массивов чисел – сначала складывались самые маленькие, чтобы накопить хоть какое-то значимое число, потом прибавлялись все больше и больше.» Опечатка в слове суммирования.

Reply

Июль 11, 2015 at 18:09
Гарик says:

```
public class AddOneFirst
{
public static void main(String[] arg) {
int x = 10;
int a = x++;
System.out.println(«x=» + x);
System.out.println(«a=» + a);
}
}
```

Как видим место расположения оператора ++ влияет на результат. В первом случае мы СНАЧАЛА ПРИСВОИЛИ переменной “a” значение из переменной “x” и только потом увеличили “x”.
Во втором случае мы СНАЧАЛА УВЕЛИЧИЛИ переменную “x” а потом присвоили ее значение переменной “a”.

Я не понял ничего можете поподробнее обяснить?

Reply

- 

  Июль 13, 2015 at 06:11
  admin says:

  Давайте на примере. Пусть a = 10. Теперь рассмотрим два варианта присваивания
  b1 = a++;
  «b1» будет равно 10, «a» будет равно 11, т.к. сначала в переменную «b1» помещается старое состояние «a» (т.е. 10). И только потом «a» увеличивается на 1

  b2 = ++a;
  b2 будет равно 11, «a» будет равно 11. В этом случае сначала «a» увеличиывается на 1 и только потом результат присваивается переменной «b2».

  Reply

- 

  Февраль 4, 2016 at 21:49
  Pavel says:

  Ромб с комментами,
  учтите что ось Y идет вниз (без матана нихрена бы не решил):

```
public class Main
{
public static void main(String[] arg) {
int size = 11;
for (int y = 0; y < size; y++) {
for (int x = 0; x < size; x++) {
if ( x == (size — 1) / 2 — y || // график 1/4 x=-y смещенный на 1/4 вправо
x == (size — 1) / 2 + y || // график 2/4 x=y смещенный на 1/4 вправо
x == y — (size — 1) / 2 || // график 3/4 x=y смещенный на 1/4 влево
x == (size — 1) / 2 — y + (size — 1) // график 4/4 x=-y смещенный на 3/4 вправо
){
```

```
System.out.print("*");
} else {
System.out.print(" ");
}
}
System.out.println();
}
}
}
```

Reply

- 

  Февраль 4, 2016 at 21:51
  Pavel says:

  Кстати писать, компилировать и выполнять ваши творения удобно в браузере:
  http://ideone.com/

  Reply

  - 

    Июнь 13, 2016 at 02:13
    WV says:

    Спасибо Дружище!
    Очень удобный инструмент для составления конспектов!
    =)

    Reply

- 

  Октябрь 2, 2017 at 19:50
  AleK says:

Такая реализация кода имеет существенные ограничения. Пробелы занимают всю область ромба при выводе результата. После символов ” * ” программа начинает отрисовывать лишние пробелы, которые по сути не нужны. Фактически количество символов в каждой строке равно количеству строк. Более того, вы пробовали ввести size = 10 ? Вобщем, если size — чётное число, программа вобще отрисовывает ромб неправильно.

Я предлагаю такую реализацию (здесь x — любое число больше 3) всвязи с чем у меня есть вопрос к автору курса.

```
class Ideone
{
public static void main(String[] arg) {
int count = 10;
for (int y = 0; y < count/2; y++) {
for (int x = 0; x < count/2+y+1; x++) {
if ( x == count/2+y ||
x == count/2-y
){
System.out.print("*");
} else {
System.out.print(" ");
}
}
System.out.println();
}
for (int y = count/2; y < count-1+count%2; y++) {
for (int x = 0; x <= count*1.5-y-2+count%2; x++) {
if ( x == count/2*2*1.5-y-(1-count%2)*2 ||
x == y-count/2+(1-count%2)*2
){
System.out.print("*");
} else {
System.out.print(" ");
}
}
System.out.println();
}
}
}
```

Пожалуйста посмотрите внимательно на строчку “x == count/2*2*1.5-y-(1-count%2)*2”. В частности на “count/2*2”. Выглядит как масло масляное, но программа тем не менее работает корректно только в случае, если вычисление этой строки мы начинаем строго с деления.

Только для этого сюда вставлена эта операция. Даже если поменять местами множитель с делителем, то есть написать — “count*2/2”, ромб сразу теряет сторону. Я сначала предполагал, что это из-за того что x — int, однако же в первом цикле где “x < count/2+y+1” — это вопросов не вызывает. Что у Java есть какая-то принципиальная разница в реализации алгоритмов умножения и деления?

Reply

- Июнь 13, 2016 at 01:54
WV says:

Приветствую mr. Admin!
Цитата: «Если вы задумаетесь о сдаче сертификата — там много коварных вопросов связанных с этим оператором. Попробуйте угадать результат следующих операторов:»
Результат: c=8 (наперво выполняется действие, как если бы оно было заключено в скобки (12-5) и далее происходит первый инкремент (7+1), затем результат передается в переменную c (8) и выполняется второй инкремент (9) — последний результат теряется)
Результат: d=12 (наперво выполняется инкремент (5+1) и далее происходит умножение (6*2), полученный результат заносится в переменную)

Насколько мы близки к истине?

Reply

  - Июнь 14, 2016 at 16:15
admin says:

Может лучше самому написать программу и проверить ?

Reply

- Июнь 13, 2016 at 04:15
WV says:

Цитата: «Именно так — т.к. переменная «a» определяет заранее результат, то часть (++b > 0) даже не проверяется. Существует возможность заставить все-таки вычислять и вторую часть. Для этого надо использовать вместо «||» выражение «|».»

А разве не достаточно обменять местами правую и левую части выражения(?): (++b > 0) || a, так как результат слева (в скобках) заранее неизвестен, то и производится вычисление (принцип короткости утрачивается объявленная переменная A находится в правой части).
Или я что-то упустил из виду?

Reply

- 
  Июнь 14, 2016 at 16:17
  admin says:

  Цель была показать, что может произойти при неаккуратном использовании булевых выражений.

  Reply

- 
  Ноябрь 16, 2016 at 10:23
  Жанна says:

  Вас очень интересно читать. Вы доступно и терпеливо объясняете. Решила обновить свои познания в Java, у Вас нашла некоторые интересные дополнения, к уже мне известным.

  Reply

  - 
    Ноябрь 16, 2016 at 12:50
    admin says:

    Спасибо. Заходите почаще.

    Reply

- 
  Март 24, 2017 at 00:28

iamke says:

Добрый вечер , подскажите, а где можно посмотреть задачки(задания,примеры) какие-нибудь не сложные, для начинающих.Спасибо заранее.

Reply

- 
  Март 24, 2017 at 09:45
  admin says:

  Очень сложно ответить на этот вопрос — хорошие задачи достаточно сложно найти. Большинство учебников хоть и пытаются что-то придумать. но получается на мой взгляд достаточно плохо. Не очень интересные задачи — и мотивация пропадает.
  Мы на наших курсах даем всякие задачи (достаточно жизненные и потому в основном их интересно решать), но это уже ноу-хау и просто так раздавать пока нет желания.

  Reply

  - 
    Март 24, 2017 at 09:59
    iamke says:

    понятно. Спасибо за ответ.

    Reply

- 
  Май 4, 2017 at 18:17
  Sasha says:

  Сайт супер

  Reply

Май 26, 2017 at 13:53
Oleg says:

Спасиба! Всё просто и понятно. Читаем дальше…

Reply

Декабрь 5, 2017 at 15:56
Михаил says:

Это ужасно!
Автор, вы о чем пишите? Вы что не понимаете, что в программировании важно каждое слово, каждое понятие и определение, каждый знак. И что все это нужно подробно, последовательно и ОДНОЗНАЧНО объяснять? А вы постоянно кидаете какие-то общие фразы и скачете через промежуточные выводы или определения. Например, что это за «(обратите внимание, что файл теперь должен называться SecondEmpty.java)»? Какой файл? Почему он должен так называться?

Или примеры программ с булевыми переменными:

```
public class ShortBool
{
public static void main(String[] arg) {
boolean a = true;
int b = 0;

boolean result = a || (++b > 0);
System.out.println(«B=» + b);
}
}
```

Почему здесь результат ноль? Если выводится значение целочисленной переменной b, которая в соответствии с оператором ++b должна иметь значение 1? Если речь о булевой result, то она тоже 1 т.к. a=true, т.е. 1. Но при чем и зачем здесь вообще нужна булева переменная result, если выводится просто число?
Далее: что значит «Для этого надо использовать вместо «||» выражение «|»»? Что делает это выражение? И почему тогда «Здесь выражение для вычисления переменной «result» выглядит немного иначе. И результат уже будет … 1.»? ПОЧЕМУ? Опять же при чем здесь булева переменная

result если сами же пишите «Так вот попробуйте угадать, какое значение переменной «b» будет выведено ?»
Вы с самого начала только запутываете новичков, вместо последовательного и не допускающего неоднозначных толкований изложения материала.

Reply

- 
  Декабрь 5, 2017 at 16:20
  admin says:

  Добрый день, Михаил. Сколь эмоционально Ваше сообщение 🙂
  Давайте начнем с названия раздела — «Укороченное вычисление логических операций». Еще раз ВНИМАТЕЛЬНО прочтите эту надпись — «УКОРОЧЕННОЕ вычисление логических операций».

  Почему же все-таки в первом случае b == 0 — а это именно так — Вы можете проверить. Снова внимательно читаем чуть выше перед указанным примером:
  «Здесь можно видеть, что для вычисления переменной «d» совсем необязательно вычислять все — операция || указывает, что в данном случае достаточно того, что переменная «a» равна «true». Все остальное можно не вычислять. Это и есть принцип укороченного вычисления.»

  Если Вы немного порассуждаете (а именно это и предлагается — порассуждать), то поймете — вторая часть после || НЕ ВЫЧИСЛЯЕТСЯ. Совсем. Эта часть не будет выполняться. Соответственно переменная «b» не изменится.

  Может быть и можно что-то добавить или как-то иначе выразить свою мысль. но всю необходимую информацию я написал. А вот дальше … это как раз проблемы «книге нельзя задать вопрос» и «книга всегда рассказывает одинаково одними и теми же словами». Увы, иногда язык повествования для кого-то может не подойти. Такое бывает. Может я что-то и придумаю, почитав комментарии.

  Reply

- 
  Март 27, 2018 at 21:48
  Максим says:

  Добрый день. Пытаюсь скомпилировать вторую программу, но при попытке в командной строке перейти к папке C:\JavaLesson выдается ошибка: «C:\JavaLeeson is not recognized as an internal or external command…». Такая же ошибка при попытке перейти к любой другой папке. Хотя вчера при компилировании первой программы, такой ошибки не возникло.

Подскажите, в чем может быть проблема?
Заранее спасибо!

Reply

- 
  Март 27, 2018 at 21:54
  Максим says:

  Нашел. Я находился в другой папке, и для перехода из нее, нужно использовать команду cd.

  Reply

Leave a reply

Comment

You may use these HTML tags and attributes: `<a href="" title=""> <abbr title=""> <acronym title=""> <b> <blockquote cite=""> <cite> <code class="" title="" data-url=""> <del datetime=""> <em> <i> <q cite=""> <s> <strike> <strong> <pre class="" title="" data-url=""> <span class="" title="" data-url="">`

Имя *

E-mail *

Сайт

− два = 6

Add comment

Designed by Blog templates, thanks to: Free WordPress themes for photographers, LizardThemes.com and Free WordPress real estate themes