

Модуль 1. Основы программирования

Тема 1.2. Типы данных и арифметические операции

2 часа

Оглавление

[1.2. Типы данных и арифметические операции](#)

[1.2.1. Программа A + B](#)

[1.2.3. Автоматическое тестирование](#)

[1.2.4. Арифметические операции](#)

[1.2.5. Операции с присваиванием](#)

[Задание 1.2.1](#)

[Благодарности](#)

1.2. Типы данных и арифметические операции

Первые электронно-вычислительные машины создавались для того, чтобы производить сложные вычисления. Сейчас компьютеры используют в самых различных областях. Это и просмотр видео, и компьютерные игры, и обработка звука. Однако в основе работы компьютеров по-прежнему остаются именно вычисления. Изображение и звук появляются благодаря размещению чисел в ячейках памяти звуковой и видекарты. 3D-эффект в играх – это результат обработки чисел алгоритмами вычислительной геометрии и т.д.

Отдельным типом данных можно (а в Java, нужно) считать строки, но даже строки для компьютера – это на самом деле последовательности чисел - кодов символов. Например, в языке Си практически нет разницы между числами и символами, она проявляется в основном только при выводе. Поэтому очень важно понимать, как устроена в языке работа с числами.

Все типы данных в Java разделяются на два класса – примитивные и ссылки на объекты. На этом занятии мы будем рассматривать в основном только числовые примитивные типы.

Обычно алгоритм получает данные, размещает их в своей памяти, обрабатывает и возвращает результат. Функция **main** главного класса имеет тип **void**, то есть фактически ничего не возвращает, но мы все равно получаем результат используя вывод.

Соответственно, для того, чтобы написать минимально полезную программу нужно

- знать, как организовать ячейки памяти для хранения данных, **заводить** так называемые **переменные** и **выполнять вычисления с ними**.
- **вводить и выводить значения переменных (в простейшем случае с консоли и на консоль)**

Для того, чтобы завести переменную, можно в любом месте функции конструкцию

<тип> <имя переменной> [= <значение>] ;

Это так называемый псевдокод. В реальных программах слова **в угловых скобках** и **сами угловые скобки не пишутся**, а вместо них употребляются соответствующие конструкции языка. **Квадратные скобки также не пишутся**. Они означают, что данная часть кода не обязательна. Например, в программе создание переменной может выглядеть так:

```
int a2;
```

или так:

```
double xx = .15;
```

(дробные числа в коде записываются через точку, при этом нулевая целая часть может опускаться).

Переменные могут называться любой последовательностью латинских букв, цифр и знаков подчеркивания при этом не могут начинаться с цифры. Лучше всего в программах давать переменным «говорящие» имена. Например, переменную, в которой хранится ширина, стоит называть `width`. Старайтесь использовать именно английские названия. Транслит в программах выглядит плохо.

Основные числовые типы – `int` – целые числа и `double` – вещественные.

Вывод чисел выполняется так же, как и строк, при помощи `PrintStream`, например

```
int x = 5, y = 7;
out.println(x + y);
```

выведет 12.

Для ввода переменных удобно использовать класс `Scanner`, вернее объект класса `Scanner`.

Создать инструмент ввода можно там же, где и инструмент вывода и по тем же причинам сделать его статическим.

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

Для ввода целых чисел, нужно использовать функцию класса `Scanner nextInt()`, например

```
int x = in.nextInt();
```

Для чтения вещественных чисел можно использовать функцию `nextDouble()`, но перед ее использованием нужно настроить `Scanner` на использование десятичной точки (а не запятой по обычному умолчанию), например вот так:

```
in.useLocale(Locale.US);
```

Это нужно сделать всего один раз, лучше в самом начале функции `main`.

Вывод вещественных чисел происходит как обычно.

Чтобы посчитать площадь круга с данным нецелым радиусом, можно написать:

```
in.useLocale(Locale.US);
//...
double r = in.nextDouble();
out.println(3.1415 * r * r);
```

При вводе вещественных чисел дробную часть после точки можно опускать. То есть будет восприниматься корректно как 35.0 так и просто 35. (с точкой).

У вещественных чисел нет жестких ограничений на величину, как у типа `int`, но при работе с ними возникает погрешность, поэтому нужно быть осторожным.

Заметим, что при использовании `Scanner`'а могут возникать исключения. Если ввести некорректные данные, например `14..a5`, то программа закончится аварийно, потому что `Scanner` не смог найти число в переданной ему информации. На самом деле, этой ситуации можно избежать. В реальной программе следует перехватить исключение и вместо аварийного завершения программы сообщить пользователю о его ошибке и попросить ввести число еще раз.

1.2.1. Программа A + B

Для примера напишем программу, которая складывает два данных ей целых числа. Сразу приведем полный текст решения, а потом разберем его.

```
import java.io.PrintStream;
import java.util.Scanner;

public class MyProgram
{
    //      public      необходим,      чтобы      Android      приложение
    // могло изменить значение переменных.
    public static Scanner in = new Scanner(System.in);
    public static PrintStream out = System.out;

    public static void main(String[] args)
    {
        int a, b, c;
        out.println("Введите два числа:");
        a = in.nextInt();
        b = in.nextInt();
        c = a + b;
        out.print("Сумма: ");
        out.print(c);
    }
}
```

Можно объединить объявление переменных с чтением:

```
int a = in.nextInt(), b = in.nextInt();
```

Но тогда естественно эти строки писать после вывода строки «Введите два числа».

Последние строки тоже можно объединить, и даже обойтись без переменной `c`

```
out.println("Сумма: " + (a + b));
```

Возникает два вопроса. Почему после строки стоит знак плюс (во многих языках – запятая) и зачем внутренние скобки?

У класса `PrintStream` есть методы вывода всех примитивных типов и еще один метод для вывода строк. Все объекты могут «представляться» строками автоматически. (Мы обсудим это позднее). И для строк (и только для них, для других классов – нет) возможно сложение с другими примитивными типами и строками. При этом происходит простое «склеивание», «конкатенация». Например, результат операции `"x" + 1` - строка `"x1"`. Таким образом, в нашем примере формируется одна строка и она выводится.

Ответ на второй вопрос – так называемая «ассоциативность операций».

Все бинарные операции (операции, где участвуют две величины, например, умножение), за исключением операторов присваивания, **левоассоциативны**.

`a - b - c` вычисляется как `(a - b) - c`, то есть «как в математике».

Операторы присваивания правоассоциативны.

Это означает что, `a = b = c` равнозначно `a = (b = c)`, что позволяет объединять операции присваивания в цепочку.

Как и на прошлом занятии, эту программу можно запустить на устройстве.

Запуск на устройстве

На мобильном устройстве программу можно запустить точно так же, как и на предыдущем занятии. Достаточно заменить код класса `Test` пакета `myprogram`. При этом оставить (или добавить к нашему коду) строки `package` и `import`. Можно и сразу писать программу в `TestBed`, не создавая нового проекта и пользоваться различными конфигурациями запуска, как описано в конце предыдущего занятия.

Работа этой программы на устройстве отличается тем, что по команде чтения `Scanner`’у показывается поле ввода, в котором требуется ввести переменную и нажать на `Enter`.

1.2.2. Автоматическое тестирование

Когда программа достаточно сложна, проверить ее правильность «глазами» очень затруднительно. Обычно в этом случае прибегают к автоматическому тестированию.

Мы знаем, как должна вести себя программа, если ей ввести определенные данные. Например, наша программа должна выдать 12, если ей сообщить 5 и 7.

В большинстве случаев для проверки достаточно сравнить полученные данные с эталонными. Эту проверку несложно поручить программе.

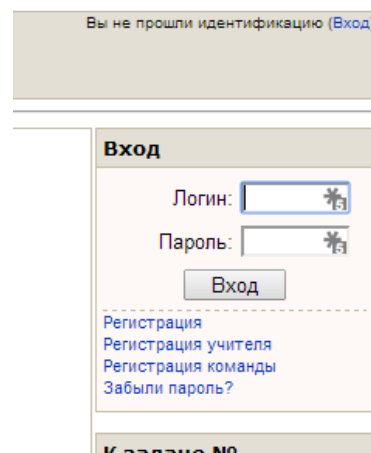
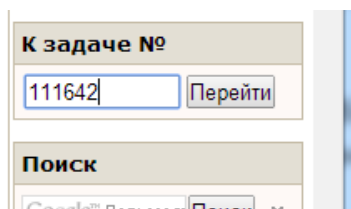
Например, на олимпиадах по программированию проверка осуществляется именно так. Участник посылает текст решения в систему, она компилирует его, запускает на наборе тестов и выдает вердикт.

Одним из ведущих сайтов в области школьной олимпиадной информатики в России является <http://informatics.msk.ru/>. Им очень удобно пользоваться для проверки своих решений по построению алгоритмов.

Для полноценной работы с сайтом <http://informatics.msk.ru/>, нужно зарегистрироваться по ссылке в правом верхнем углу страницы.

После этого в правой колонке можно указать номер задачи.

Задача «А + В» имеет номер **111642**



Обратим внимание на ее условие.

Входные данные

На первой строке входного файла находятся два целых числа a и b ($-10^9 \leq a, b \leq 10^9$).

Выходные данные

Вашей программе требуется вывести единственное число — сумму заданных чисел $a + b$.

Условие в скобках писать в программу не нужно. Это так называемое ограничение входных данных. Оно говорит, что программа должна корректно работать, если введенные данные соответствуют этому условию. При этом программа может как угодно работать или даже аварийно завершаться, если входные данные ему не соответствуют. Если «пользователь» введет дробные числа или, например, три числа или даже свое ФИО, программа может вести себя любым образом.

В олимпиадных задачах входные данные всегда корректны, то есть точно соответствуют условию и примерам.

Это очень важное правило. Оно дает возможность участнику сосредоточиться именно на разработке алгоритма, а не строить вместо этого совершенную «защиту от дурака», что, наоборот, абсолютно необходимо делать при разработке прикладных программ.

Программа не должна выводить лишнего. Это будет расценено системой как неверный ответ. Нужно убрать из программы строки с выводом приглашения и строки «Сумма:»

Перед посылкой надо проверить на тестах из условия.

2 3 получается 5.

17 -18 выводится – 1

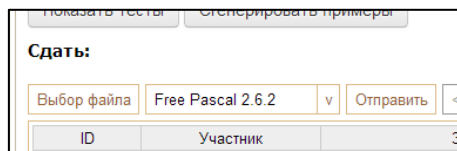
Технически informatics требует, чтобы программа была в одном файле, в пакете по умолчанию. Поэтому package нужно тоже закомментировать.

Таким образом, файл-решение для этой задачи должен быть таким:

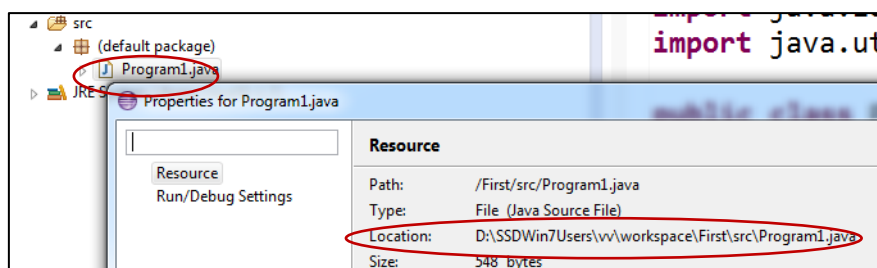
```
// package myprogram;
// import ru.samsung.itschool.testbed.*;
import java.io.PrintStream;
import java.util.Scanner;

public class Test
{
    static Scanner      in = new Scanner(System.in);
    public static PrintStream out = System.out;
    public static void main(String[] args)
    {
        int a, b, c;
        a = in.nextInt();
        b = in.nextInt();
        c = a + b;
        out.print(c);
    }
}
```

Под условием есть раздел «Сдать».



Месторасположение исходного файла (он находится в папке src проекта в папке Workspace, нужно лишь знать расположение папки Workspace), его можно определить при помощи щелчка правой кнопки мыши по имени файла ⇒ Properties в Навигаторе.



В графе статус написано «ОК».

Участник	Задача	Дата	Язык	Статус	Пройдено тестов	Баллы	Подробнее
Владимир Ильин	111642. Сумма двух чисел	2014-06-10 09:42:40	Java JDK 1.7	OK	19	100	Подробнее
Владимир	111642. Сумма	2014-04-12	Java				

Это бывает не всегда. В случае ошибок программа проходит не все тесты, можно нажать ссылку подробнее, понять, в чем ошибка и перепослать.

1.2.3. Арифметические операции

С вычислениями в Java все достаточно стандартно. Можно использовать четыре арифметические операции (умножение записывается звездочкой (*), а деление прямым слесом (/). Приоритеты операций стандартны.

Но при программировании на Java нужно быть очень внимательным, учитывать особенности арифметических операций.

- Арифметические операции могут вызвать переполнение.

Целые числа в Java (переменные типа int) не могут содержать значения большие 2^{31} по модулю. Это чуть больше двух миллиардов. Когда результат операции превышает этот предел возникает переполнение – ответ становится неверным.

Обязательно посмотрите, что выведет фрагмент:

```
int x = 1000 * 1000;  
out.println(x * x);
```

Целые числа в Java (переменные типа int) не могут содержать значения большие 2^{31} по модулю. Это чуть больше двух миллиардов.

- Деление целых чисел выполняется нацело.

Например, результат **5 / 3 равен 1**.

- Операция процент (%) позволяет вычислить остаток от деления.

Например, **5 % 3** это **2**. С отрицательными числами остаток от деления работает не так, как подсказывает интуиция. Проверьте, что выведет строка:

```
out.println((-7) % 3);
```

и поймите, почему получается именно такой результат.

Операция взятия остатка очень часто используется в программировании. Например, проверку четности числа удобно делать так:

```
/*сравнение в Java выполняется удвоенным знаком равенства (==)*/  
if (a % 2 == 0)...
```

Операция % применима в Java только к целым числам.

- В Java нет операции возведения в степень!
- Крышечка (^) работает не как степень (в Java это логическая операция)

Чтобы операция деления производилась без отсечения дробной части (обычное деление, ненацело) нужно, чтобы одно из чисел было нецелым (либо вещественной константой, либо переменной нецелого типа).

Например, если

```
int a = 7, b = 8;
```

тогда команда

```
out.println((a + b) / 2);
```

выведет 7, а команда

```
out.println((a + b) / 2.0);
```

выведет 7.5

1.2.4. Операции с присваиванием

Чтобы изменить переменную, скажем, получить показания часов через минуту, обычно пишут примерно такой код:

```
minutes = minutes + 1;
```

Однако в Java есть более удобные средства.

Можно записать:

```
minutes += 1;
```

Существуют комбинированные операции с равенством со всеми основными операциями. То есть возможно использовать `--` `/=` `%=` и так далее. Нужно стараться использовать именно эти операции. Они выполняются быстрее, это понятнее и проще читается.

Если увеличение или уменьшение должно быть именно на 1, то можно использовать операции `++` (инкремент) и `--` (декремент), например

```
minutes++;
```



В предыдущей строке приведена **постфиксная** форма операции инкремента. Существует также **префиксная** форма инкремента и декремента.

```
++minutes;
```

Результат от этого не изменится.

Рассмотрим другой пример:

```
1. saveMinutes = minutes++;
```

Это не эквивалентно

```
2. saveMinutes = ++minutes;
```

В данном случае результат выполнения будет разным!

В первом случае сначала в переменной saveMinutes будет сохранено значение minutes, и только потом значение minutes будет увеличено на 1. Во втором случае сначала значение minutes будет увеличено на 1, а потом результат будет присвоен переменной saveMinutes. Таким образом во втором случае в переменной saveMinutes будет сохранено значение на 1 больше, чем в первом случае.

Таким образом, при префиксной форме инкремента/декремента сначала выполняется увеличение/уменьшение на 1, а затем все остальные операции в выражении. При постфиксной - сначала все операции в выражении, а затем увеличение/уменьшение на 1.

Важно знать об этой разнице, но, начинающим программистам не стоит использовать инкремент и декремент в одном выражении с присваиванием. Если этого не делать, результат обеих форм будет одинаковый.

Задание 1.2.1

Решите (сдайте на ОК) задачи на informatics №№ 2941, 2942, 2944, 2945, 2947.

При решении этих задач **не** следует использовать условные конструкции.

Благодарности

Компания Samsung Electronics выражает благодарность за участие в подготовке данного материала преподавателю ИТ ШКОЛЫ SAMSUNG Ильину Владимиру Владимировичу.