Занятие №1. Знакомство

Предполагается, что вы знакомы с языком Java.

Прежде всего, с синтаксисом этого языка. Представляете себе, как работать в среде программирования: компилировать и запускать программы. (Мы будем использовать Eclipse, хотя это абсолютно непринципиально, кроме, возможно, второго занятия, в котором речь пойдёт об отладчике.)

Если это не так, но вы имели опыт программирования на других языках, то быстро освоитесь с помощью Справочника, который приведён в конце пособия или литературы. Можем порекомендовать книгу «Java 2» авторов Кея С. Хорстманна и Гари Корнелла (в двух томах), которая выдержала много переизданий. Для быстрого старта достаточно ознакомиться только с третьей главой первого тома.

Алгоритмы

Мы будем говорить об алгоритмах, том моторе, на котором работают программы и который практически всегда скрыт от пользователя. Word распределяет промежутки между символами так, чтобы строки заполнялись равномерно, Excel вычисляет значение выражения с учётом скобок, но как именно они это делают? В изучении алгоритмического программирования мы заглянем «внутрь» программ! Оказывается, что «внутри» программы выглядят даже интереснее, чем «снаружи».

Прежде всего, о том, что будет не похоже на ваш прежний программистский опыт.

Проще сказать, чего в этом курсе не будет. Вам не надо будет заботиться об ошибках пользователя и организации для него удобного интерфейса, что занимает порой более половины времени разработки программ. Для нас будут существовать только «входные» и «выходные» данные — написанные в текстовом виде числа и строки. Из всех средств интерфейса останется только примитивный ввод-вывод в файл или на консоль. Для тех, кто забыл, как это делается, в конце занятия разбирается простейшая задача «A+B».

Мы не будем давать чёткого определения алгоритма, тем более что дать его достаточно сложно, а поговорим на примерах.

Сколько проверок на делимость надо осуществить, чтобы посчитать количество делителей у числа 1200567? Если вы скажете, что 1200566, «просто проверим все числа меньшие данного», то будете неправы. Правда, зачем проверять, делится ли наше число на 900000? Конечно, не делится! Приходит в голову мысль: проверять нужно только половину! Вот мы уже сократили время работы нашего алгоритма вдвое. А зачем

проверять делимость на четыре, восемь и так далее, если мы уже выяснили, что на два наше число не делится? Еще в два раза выигрыш по времени! Но оказывается, можно сократить ещё, и гораздо сильнее! Как? Вы узнаете об этом на третьем занятии.

А теперь попробуем оптимизировать расход оперативной памяти. Пусть вам диктуют длинную последовательность чисел. А у вас только маленький листочек, на котором с трудом умещаются только два числа, карандаш и ластик. Известно, что в этой последовательности одно число повторяется больше других, вместе взятых. Как вам действовать, чтобы безошибочно указать это число, когда диктовка закончится? Назвать первое или последнее, авось попадем? Ненадежно, ведь последовательность может быть 1 2 2 2 3 2 3. И нужно назвать 2 в этом случае... Это достаточно сложная задача, попробуйте решить её сами или обратитесь к Подсказке 1.1.

А мы начнём с совсем простой задачи.

Найти максимум из трёх заданных чисел. Сложного ничего вроде нет. Просто сделать много сравнений, разобрать все случаи... В том и проблема, что *все* случаи разобрать, причём правильно, достаточно сложно.

Попробуйте быстро найти ошибку в этой программе:

```
if (a > b)                                  // Программа выдает
{                                           // неверный ответ на входе:
     if (b > c) System.out.println(a);// a:___ b:___ c:___
     else System.out.println(c);
}
else
{
     if (a > c) System.out.println(b);
     else System.out.println(c);
}
```

Что надо ввести, чтобы программа выдала неправильный ответ?

Предполагается, что вы будете работать с этой книгой с карандашом. Если вы видите в листинге подчеркивание, значит, его надо заполнить самостоятельно.

В программах этого занятия для «быстрого старта» мы организуем вывод на консоль стандартный Java-объект `System.out`*. В Справочнике подробно описаны несколько способов ввода-вывода и в программах следующих занятий используется просто объект out, который в принципе может быть представителем разных классов, в частности выводить данные в файл.*

Можно использовать логическую функцию &&, чтобы упростить код, записав понятие максимума «напрямую»:

```
if (a > b && a > c) System.out.println(a);
```

```
if (b > a && b > c) System.out.println(b);
if (c > a && c > b) System.out.println(c);
```

Но и эта программа, оказывается, даёт неверный результат, например, на всех равных числах. Причём если заменить все знаки «больше» на «больше или равно», лучше не становится — программа начинает выдавать ответ по три раза...

Хорошее решение состоит в том, чтобы «свести задачу к предыдущей» — к максимуму из двух чисел! Найдём максимум из a и b, а потом из этого максимума и c:

```
int max = a;
if (max < b) max = b;
if (max < c) max = c;
System.out.println(max);
```

А если мы решение задачи «максимум из двух чисел» оформили бы в виде функции, решение этой задачи можно было бы записать в одну строку:

```
System.out.println(max(max(a, b), c));
```

При этом строку абсолютно очевидную!

И такое решение можно масштабировать на любое количество переменных практически без уменьшения понятности. Что мы сделали? Придумали другой алгоритм и красиво его записали. От этого программа стала короткой и правильной.

И всё же, как проверить, что программа работает верно? Абсолютно точно выяснить это в сложных случаях... практически никак нельзя! Потому что надо «перепробовать» все возможные варианты, а их может быть очень много, часто бесконечно.

Но получить уверенность в правильности программы-решения, можно, ведь **принципиально разных** случаев не так уж и много. Например, для нашей задачи случаи 1 2 2 и 3 5 5 практически одинаковы. Если наша программа работает правильно на первом наборе данных – говорят «на первом тесте» – , то, скорее всего, работает правильно и на втором. Сколько принципиально различных случаев, например, в программе «максимум из трёх»?

Но, все равно, случаев бывает много, обычно несколько десятков. Проверять их вручную долго и ненужно, обычно пользуются для этого специальным программным комплексом – тестирующей системой.

Тестирующая система

Составители задачи по алгоритмическому программированию, будем называть их в дальнейшем «жюри», тщательно разрабатывают систему тестов — наборы входных

данных, охватывающие, по возможности, все принципиально разные случаи. Например, для задачи «максимум из двух чисел» таких наборов должно быть не меньше трёх:

- первое число больше второго;
- второе больше первого;
- числа равны;

Почему «не меньше»? Предположим, что участник решил находить максимум... делением. Абсурд? Вовсе нет, ведь деление в целых числах в Java выполняется нацело!

```
System.out.println((a / b) * a + (a / b) * b);
```

Красиво! В одну строку, вообще без «если» и работает всегда.

…Почти всегда... В систему добавляется ещё один тест типа «4 0».

Дальше всё просто. Специальная программа — тестирующая система — вместо человека запускает программу-решение, передавая ей в качестве данных разработанные тесты. И оценивает вывод программы. Часто, но не всегда, для этого достаточно сравнить ответ, выданный тестируемой программой, с эталонным, полученным заранее. Например, для ввода 1 2 наша программа должна обязательно выдать 2. Иногда делается что-то более интеллектуальное. Например, требуется разложить число на сумму слагаемых. Тестирующей системой запускается специальная программа-«чекер», которая берёт выходные данные из программы решения и складывает их, вычисляя, составляют ли полученные числа требуемую сумму.

На некоторых соревнованиях самим участникам предлагается придумывать тесты, на которых программа соперника будет работать неправильно .Это действие называется «взлом». Участники в этой фазе соревнования действуют как настоящие хакеры — используют «уязвимости» в чужих программах.

Естественно, придуманные тесты должны полностью подходить под ограничения, указанные в условии.

В зависимости от подхода (правил соревнований, желания преподавателя) тестирующая система либо запускает наше решение в любом случае на всех тестах, и мы получаем баллы за каждый пройденный тест, либо проверяет до первого «провала», и мы получаем сообщение, что задача пока не решена.

В графе «Результат» указывается результат проверки. Он может быть:

- **OK** — программа прошла все тесты, решение верное.
- **Wrong Answer (Неправильный ответ, WA) —** программа прошла не все тесты, то есть работает не во всех случаях. В этом случае в графе «Ошибка на тесте» показывается номер теста, на котором программа выдаёт неверный ответ.
- **Presentation Error (Неправильный формат вывода, PE) —** означает, что на каком-то тесте программа выводит ответ не в том формате, как это требуется в условии задачи (например, выводит несколько чисел, когда требуется одно, или выводит слово, когда требуется число).
- **Runtime Error (Ошибка выполненияRE) —** означает, что на каком-то тесте программа выполняет недопустимую операцию (например, происходит деление

на 0, выход за пределы массива или иная ошибка, которая может привести к аварийному завершению программы).

- **Memory Limit (Превышен предел по памяти, ML) —** превышен максимальный объём памяти, выделяемый для программы по условию.
- **Time Limit (Превышен предел времени выполнения программы, TL) —** программа работает слишком долго.
- **Compile Error (Ошибка компиляции, CE) —** означает, что программа содержит синтаксические ошибки из-за чего тестирующая система не способна её откомпилировать и запустить на проверку. Часто возникает из-за попытки отправить задачу на другом языке программирования.

С одной стороны, это очень удобно: мы знаем результат проверки буквально через несколько секунд после посылки решения. С другой — использование тестирующей системы предполагает чёткое следование правилам оформления решений:

Программа-решение должна чётко соблюдать формат входных и выходных данных. Никаких «Введите стороны прямоугольника» или «Площадь равна:». Всё это будет считано чекером как вывод программы и воспринято как неверный ответ. Нужно выводить данные именно так, как указано в условии, и обычно ещё в примере.

Программа-решение должна отработать (закончить выполнение) быстро. Не надо делать задержек, чтобы «посмотреть», что вывела программа. Тестирующая система воспримет это как неэффективный по времени работы алгоритм и прервёт его работу.

Ещё очень важным правилом является то, что тестирующая система не ошибается как пользователь человек и может передавать данные без ошибок, всегда строго соблюдая формат.

Входные данные всегда корректны. При решении задач вам нет необходимости проверять, что пользователь введёт «что-то не то». Обычно в условии чётко сказано, каким ограничениям соответствуют входные данные, и система тестов им всегда должна соответствовать. Например, если сказано, что вводится натуральное число до 100, — вам нет нужды описывать в программе случаи, когда «пользователь» введет ноль, минус один или 125.7 — в этом случае ваша программа может работать как угодно: выводить неверный ответ, зависать или даже «ломаться» — вылетать с ошибкой. Таких случаев точно при проверке не будет. Из этого следует важное правило: **ограничения на входные данные учитываются при разработке программы, но никак не записываются в программу**.

В заключение рассмотрим условие и решение задачи, без которой не обходится практически ни одна олимпиада. На пробном туре очень часто для проверки работоспособности системы предлагается решить задачу «A+B».

Задача «A+B»

Условие задачи

Даны 2 целых числа: A и B. Требуется вычислить их сумму.

Ограничения по времени и памяти

16 мегабайт,

1 секунда на каждом тесте.

Входные данные

Во входном потоке в единственной строке через пробел записаны два целых числа: A и B.$-10^9 \leq A, B \leq 10^9$

Выходные данные

В выходной поток следует записать единственное целое число — сумму чисел A и B.

Пример

Исходные данные	Результат
2 2	4
3 2	5

Решение (чтение и вывод на консоль)

```
import java.io.*;
import java.util.*;

public class Sum
{
     public static void main(String[] args)
     {
          Scanner in = new Scanner(System.in);
          PrintWriter out = new PrintWriter(System.out);

          int a = in.nextInt();
          int b = in.nextInt();
          out.println(a + b);

          out.flush();
     }
}
```

Технические моменты

В функции `main` прежде всего создаются два объекта классов `Scanner` и `PrintWriter`, для операций ввода и вывода.

У класса Scanner есть много функций для ввода различных типов данных. Они приведены в Справочнике. Например, в программе, решающей «A+B» мы используем `nextInt()` для чтения целочисленных значений `a` и `b`.

Далее мы выводим ответ. Для вывода достаточно функций `println` и `print` класса `PrintWriter` (вывод с переводом строки и без него).

В конце программы нужно закрыть объект `Scanner` командой close и обязательно выполнить команду `flush` для `PrintWriter`-а,чтобы очистить буфер объекта, и данные появились на экране или в файле.

Немного о логике решения

В тексте нет лишнего вывода `out.println("Введите два числа a и b")` или `out.print("Сумма равна").`

После вывода ответа программа мгновенно закрывается. Если мы запустим ее не в Eclipse, а например, через .bat-файл со строкой `java Sum`, то после ввода окно сразу закроется и пользователь ничего не увидит. Но с точки зрения тестирующей системы это как раз работает правильно.

Ограничение условия $-10^9 \leq A, B \leq 10^9$ учитывается (для данных используется тип `int`, подробнее мы поговорим об этом на следующем занятии), но не записывается в программе.

Ограничения по времени и памяти в этой задаче значительно превосходят необходимые. Памяти нам нужно всего пару ячеек, нет большого объёма вычислений. Поэтому в данном случае на них можно не обращать особого внимания.

Перед посылкой в тестирующую систему нужно обязательно проверить работоспособность программы хотя бы на тестах из условия. То есть ввести 2 2 и получить 4.

Подробнее о вводе и выводе на языке Java написано в Справочнике.

Немного о задачах

Задачи этого Занятия достаточно простые. Они как раз для «Знакомства». Вернее, для «воспоминания». Для того, чтобы их решить, достаточно знать только циклы и условия. Шаблон программы, простейшая программа должна все-таки содержать порядка десяти строк, можно скопировать из Справочника. Чтобы полностью освоиться с техникой ввода вывода стоит почитать разбор задачи «Следующее и предыдущее».

Все задачи подробно разбираются в Разборах, но не надо спешить туда заглядывать. Постарайтесь выработать привычку знакомиться с разбором задачи, уже после ее решения или в случае серьезных затруднений.

Для некоторых задач указаны источники – олимпиады, на которых они предлагались. Заметим, что на олимпиадах периодически предлагаются довольно простые задачи.

Задачи

Задача A. Следующее и предыдущее

Напишите программу, которая считывает целое число и выводит текст, аналогичный приведенному в примере. Пробелы, знаки препинания, заглавные и строчные буквы важны!

Пример

Ввод	Вывод
179	The next number for the number 179 is 180. The previous number for the number 179 is 178.

Задача B. Максимум из трех

Формат входных данных

Даны три целых числа, каждое записано в отдельной строке.

Формат выходных данных

Выведите наибольшее из данных чисел (программа должна вывести ровно одно целое число).

Задача C. Парты

Сколько понадобится парт, чтобы рассадить A школьников, если за одну парту можно посадить одного или двух человек? За каждой партой должен сидеть хотя бы один человек. Укажите все варианты.

Формат входных данных

Вводится одно натуральное число — A ($1 \le A \le 10000$)

Формат выходных данных

Выведите упорядоченный по возрастанию набор чисел – все возможные значения количества необходимых парт.

Примеры

Ввод	Вывод
6	3 4 5 6

Задача D. Апельсины бочками

Бизнесмен Василий после прочтения известной книги решил открыть новый бизнес – отгружать апельсины бочками. Партнерам важно знать, сколько именно бочек апельсинов отгружается каждый день.

Мобильный телефон Василия поддерживает только транслит, поэтому он передает сообщения вида "`N bochek`" . Например, "`3 bochki`" или "`1 bochka`" .

Напишите программу, которая выбирает правильное слово (из "`bochka`" , "`bochek`" , "`bochki`") в зависимости от *N*.

Формат входного файла

Одно число *N* ($0 \le N \le 1000$).

Формат выходного файла

Фраза на транслите (см. примеры).

Задача E. Четные и нечетные числа

Даны три целых числа *A*, *B*, *C*. Определить, есть ли среди них хотя бы одно четное и хотя бы одно нечетное.

Формат входных данных

Числа *A*, *B*, *C*, не превышающие по модулю 10000.

Формат выходных данных

Одна строка – "`YES`" или "`NO`".

Задача F. Кольцевая

Источник: Турнир Архимеда 2010 года

Два автомобиля движутся по кольцевой дороге длины L в противоположных направлениях. Они начинают движение из одной точки и едут с постоянными скоростями v_1 и v_2 соответственно. Требуется определить, на каком расстоянии друг от друга они окажутся в момент времени T.

Формат входных данных

На вход подаются 4 натуральных числа L, v_1, v_2, T, разделенных пробелом. Все числа не превосходят 100.

Формат выходных данных

Выведите расстояние между автомобилями в момент времени T – длину кратчайшей из двух дуг дороги между автомобилями.

Примеры

Входные данные	Выходные данные
10 1 2 1	3
10 2 3 2	0

Задача G. Квадратные таблицы

Источник: олимпиада ФМШ 2007

Вася записывает в клетки квадратной таблицы NxN натуральные числа по порядку, сначала заполняя первую строку слева направо, затем вторую и т.д. (см. рисунок слева). Петя заполняет такую же таблицу, расставляя числа сначала в первый столбец сверху вниз, затем во второй столбец и т.д.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

1	5	9	13
2	6	10	14
3	7	11	15
4	8	12	16

При этом оказалось, что некоторые числа и Вася, и Петя записали в одну и ту же клетку (например, число 6 записано во вторую строку второго столбца обеих таблиц).

Вам требуется написать программу, выводящую все числа, которые в обеих таблицах записаны в одних и тех же клетках.

Входные данные

Вводится одно число - размер таблицы.

Выходные данные

Программа должна вывести все числа, которые в обеих таблицах стоят на одном и том же месте, в порядке возрастания, через пробел.

Примеры

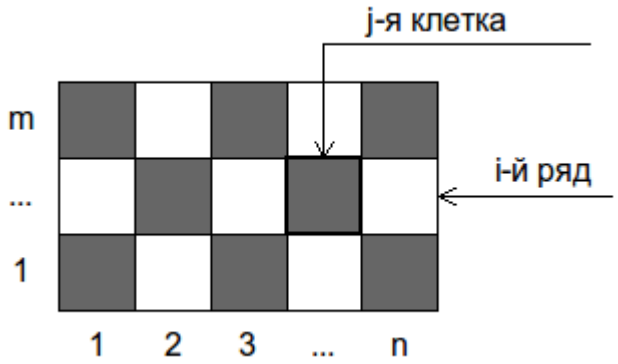
Входные данные	Выходные данные	Комментарий
4	1 6 11 16	В таблицах 4 x 4 на одних и тех же местах стоят четыре числа: 1, 6, 11, 16 (см. рисунки).
1	1	В таблице из одной клетки записано единственное число 1, оно и является ответом на вопрос задачи.

Ограничения

Размер таблицы - натуральное число, не превосходящее 100.

Задача H. Шахматная доска

Источник: Региональный этап Всероссийской олимпиады 2010-2011 года



Аня разделила доску размера m x n на клетки размера 1 x 1 и раскрасила их в черный и белый цвет в шахматном порядке. Васю заинтересовал вопрос: клеток какого цвета получилось больше — черного или белого.

Для того, чтобы выяснить это, он спросил у Ани, в какой цвет она раскрасила j-ю клетку в i-м ряду доски. По этой информации Вася попытался определить, клеток какого цвета на доске больше.

Требуется написать программу, которая по размерам доски и цвету j-й клетки в i-м ряду определит, клеток какого цвета на доске больше — черного или белого.

Формат входного файла

Входной файл содержит пять целых чисел: m, n, i, j и c ($1 \le m, n \le 109$, $1 \le i \le m$, $1 \le j \le n$, $c = 0$ или $c = 1$). Значение $c = 0$ означает, что j-я клетка в i-м ряду доски раскрашена в черный цвет, а значение $c = 1$ — в белый цвет.

Формат выходного файла

Выходной файл должен содержать одно из трех слов:

- `black`, если черных клеток на доске больше,
- `white`, если белых клеток на доске больше,
- `equal`, если черных и белых клеток на доске поровну.

Примеры входных и выходных данных

Входные данные	Выходные данные
3 5 1 1 0	black
3 5 2 1 0	white
4 4 1 1 1	equal

Задача I. Футбол

Источник: дистанционная олимпиада сайта informatics.mccme.ru

Вместо того чтобы делать уроки, Петя смотрел футбольный матч и записывал счет, который показывался на табло, после каждого забитого гола.

Например, у него могла получиться такая запись:

1:0

1:1

1:2

2:2

2:3

После этого он сложил все записанные числа: 1+0+1+1+1+2+2+2+2+3=15.

По сумме, получившейся у Васи, определите, сколько всего мячей было забито в матче.

Входные данные

Вводится одно натуральное число, не превосходящее 1000 – сумма, полученная Васей.

Выходные данные

Выведите одно число – общее количество забитых мячей.

Входные данные	**Выходные данные**
3	2
1	1

Задача J. Сумма последовательности - 2

Найдите сумму последовательности натуральных чисел, если признаком окончания конца последовательности является два подряд идущих числа 0.

Числа, следующие после двух подряд идущих нулей считывать не нужно.

Ввод	**Вывод**
1 7 0 9 0 0 5	17

Подсказки и решения. 1 Занятие.

1.1

Запишем первое продиктованное число и единицу — оно нам встретилось один раз. Встречаем записанное число — увеличиваем «счётчик» на единицу. Другое — уменьшаем. Если счётчик стал нулевым, начинаем сначала.Так как искомых чисел больше половины от всех, в конце на листочке обязательно окажется требуемое число.

1.2

Необходимо, *сначала* вычислить сумму `A + 1` *и потом* ее приписать к тексту. Иначе программа будет работать неверно, например, для введенного числа 5 выведет не 6, а 51.

1.3

Можно выводить последнее число (`A`) отдельно, после цикла.

1.4

Заметим, что это числа расположенные по диагонали

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

1	5	9	13
2	6	10	14
3	7	11	15
4	8	12	16

Для таблицы из условия (размером 4x4) это числа от 1 до 16. Они идут через 5.

1.5

Порядок важен. Первым надо считывать именно `p` оно должно стать предыдущим. На сумму это не повлияет, а на останов – да. Если перепутать порядок и ввести 0 3 0 7 0 0, то программа остановится после считывания тройки, вернее нуля, который за ней. А `s` надо инициализировать считанным до цикла `p`.

Разбор. Занятие 1

1A. Следующее и предыдущее

Эта задача на понимание формального подхода ко входным и выходным данным и базовую технику ввода вывода.

За основу программы возьмем шаблон из справочника.

```
import java.io.*;
import java.util.*;

public class Solution
{
     public static void main(String[] args)
     {
          Scanner in = new Scanner(System.in);
          PrintWriter out = new PrintWriter(System.out);

     // ... код решения конкретной задачи

          in.close();
          out.flush();
     }
}
```

Заменим комментарий нашим кодом или добавим его сразу после комментария.

Сначала нужно ввести число A. Заведем в программе целочисленную переменную A и сразу считаем в нее число.

```
A = in.nextInt();
```

Обратите внимание, несмотря на «длинный, человеческий» вывод мы не делаем «человеческого» ввода, не выводим «пользователю» никаких подсказок, что он должен ввести, а сразу читаем данные.

Теперь выведем ответ. Каждая из строк ответа состоит из четырех частей. Текст, потом число, потом текст, потом число и снова текст.

Для первой строки

`The next number for the number 179 is 180.`

первое выделенное число – это значение переменной A, а второе – значение выражения A + 1. Последняя часть – текст состоит только из одной точки.

Необходимо помнить о пробелах. Они являются частью текста. Выведем все 5 частей.

```
out.print("The next number for the number ");
out.print(A);
out.print("is ");
out.print(A + 1);
out.print(".");
```

Воспользуемся тем, что строки можно «складывать», в том числе и с числами. В этом случае они просто «приписываются» друг к другу. Все пять строчек можно заменить одной командой:

```
out.println("The next number for the number " + A + " is " + (A
+ 1) + ".");
```

Еще раз обратите внимание на пробелы и кавычки. Текст пишется в кавычках, а выражения без них.

Мы еще заменили `out.print` на `out.println` для того, чтобы он закончился переводом строки и вторая строка вывелась ниже.

Для второй строки все делается аналогично.

Мы так же могли использовать для перевода строки специальную последовательность "\n", это, кстати, дает возможность сделать вывод даже в одну команду:

```
out.print("The next number for the number " + A + " is " + (A +
1) +".\n" + "The previos number for the number " + A + " is " +
(A - 1) +".");
```

Но это получается ненаглядно, сложно читается. Поэтому лучше написать в две строки.

А зачем скобки, ведь «от перемены мест слагаемых сумма не меняется»? (Подсказка 1.2)

Таким образом, полное решение, которое мы добавляем в шаблон, может состоять из трех строк. Строки объявления переменной и ее ввода и двух строк вывода ответа out.print.

1B. Максимум из трех

Задача подробно разбирается в тексте занятия. Для решения можно использовать любой из способов.

Покажем, как записать решение при помощи функции. Напишем свою статическую функцию max:

```
static int max(int a, int b)
{
```

```
    int res = a;
    if (a > b) res = b;
    return res;
}
```

Ее надо разместить *рядом* с функцией main (внутри класса, до или после main).

Строка решения указана в тексте занятия:

```
out.print(max(max(a, b), c));
```

Отметим, что свою функцию можно и не писать, а воспользоваться стандартной статической функцией класса Math:

```
out.print(Math.max(Math.max(a, b), c));
```

1C. Парты

Для решения этой задачи требуется использовать цикл, потому что ответов может быть несколько.

Минимальное количество парт равно половине от числа учеников, если их количество четно и на единицу больше, чем «меньшая половина» от числа учеников, если нечетно (действительно, одного сажаем за отдельную парту, остальных плотно – по двое).

```
int mindesks = 0;
if (A % 2 == 0) mindesks = A / 2;
else mindesks = (A – 1) / 2 + 1;
```

Заметим, что из-за того, что деление целых чисел в Java осуществляется нацело с отбрасыванием дробной части (об этом мы будем подробно говорить на втором занятии), второе присваивание можно написать короче: `midesks = A / 2 + 1`.

Максимальное количество парт мы должны будем задействовать, если посадим каждого ученика за отдельную парту. Это число равно `A`.

Запишем цикл вывода ответа.

```
for (int i = mindesks; i <= A; i++)
{
    out.print(i + " ");
}
```

Не забывайте разделять числа пробелами при выводе. Кстати, в нашей программе после последнего значения выведется лишний пробел. Хотя мы его не увидим, этот недостаток стоит исправить. Как? (Подсказка 1.3)

1D. Апельсины бочками

Напомним пару технических моментов. Остаток от деления можно получить в Java операцией `%`. Например, последнюю цифру числа x можно получить так: `x % 10`. При записи условий нельзя использовать двойные неравенства как в математике. Вместо ~~`5 <= x <= 20`~~ нужно писать `5 <= x && x <= 20`, используя операцию `&&` - логическую И, конъюнкцию.

Логческое ИЛИ, дизъюнкция, записывается в программе на Java при помощи двух вертикальных палочек (`||`). Например, мы можем написать в программе

```
if (5 <= N && N <= 20 || N >= 25 && N <=30)
{
     out.println(N + " bochek");
}
```

Разбирать логическую сторону решения задачи не будем, иначе она потеряет свою привлекательность. Попробуйте самостоятельно учесть все возможные случаи и при этом свести их количество к минимуму.

1E. Четные и нечетные числа

Можно просто перебрать все возможные случаи четных и нечетных:

Пусть числа лежат в переменных a,b и c

```
if (a % 2 == 0 && b % 2 != 0 && c % 2 != 0 ||
a % 2 = 0 && b % 2 == 0 && c % 2 != 0 ||
. . .) out.println("YES");
else out.println("NO");
```

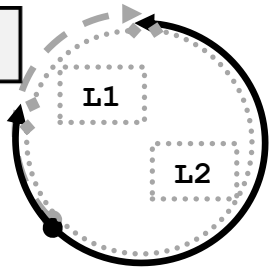
Но их очень много. Строка получается длинной и некрасивой. Легко будет допустить в ней ошибку. Интерес этой задачи в том чтобы придумать правильное и не очень длинное решение.

Один из вариантов - идти от обратного. Рассмотреть случаи "NO", а "YES" выводить по "иначе". Таких случаев всего два: *все четные или все нечетные*. Причем второй вариант можно записать совсем коротко, потому что произведение чисел нечетно только тогда, когда все они нечетны.

1F. Кольцевая

Посчитаем сколько проехали автомобили без учета полных кругов, для этого возьмем их по модулю `L` (возьмем остаток от деления на `L`).

```
S2 = L – v2 * t % L
```



```
S1 = v1 * t % L
```

Можно считать, что машины двигались в одну сторону, при этом вторая машина проехала `S2 = L - v2 * t % L`.

Модуль разности есть длина одной дуги между ними (`L1`).
Длина другой `L2` - равна `L - L1`.

Минимум из `L1` и `L2` есть ответ.

1G. Таблицы

Задача оказывается очень простой, если понять какие именно числа выводить, сколько их, и в какой они закономерности. Впрочем, данный комментарий можно отнести практически к любой задаче… Можно посмотреть подсказку 1.4.

1H Шахматная доска

Прежде всего, заметим, что если количество клеток четное, то черных и белых клеток поровну.

Если это не так, то нужно понять как «устроена» доска. Если немного порисовать и посчитать, можно увидеть, что а) клетки, у которых координаты x и y одинаковой четности одного цвета, а у которых разной – другого. б) клеток с одинаковой четностью координат – больше. Проверьте!

Таким образом, в ветке `else` можно написать

```
if ( i % 2 == 0 && j % 2 == 0|| i % 2 != 0 && j % 2 != 0)
{
     if (c == 0) out.printn("black");
     else out.println("white");
}
else
{
     // ...аналогично
}
```

Первое условие лучше заменить на `i % 2 == j % 2`
или даже на `(i + j) % 2 == 0`. Действительно, сумма двух чисел с одинаковой четностью четна!

1I. Футбол

С каждым голом в сумму добавляется текущее число голов. То есть фактически сумма, полученная Петей, это сумма 1 + 2 + 3 +…

Можно циклом while вычислять эту сумму до тех пор, пока не достигнем того числа, которое получил Петя. Количество итераций цикла (значение переменной `i`) и есть ответ.

```
// N – сумма полученная Петей
int s = 0; // текущая сумма
int k = 1; // очередное число в сумме
int i = 0; // количество итераций
while (s < N)
{
     s += k;
     k++;
     i++;
}
```

Количество итераций цикла (значение переменной **i**) и есть ответ.

Внимательно изучите, как меняются в цикле переменные **s**, **i** и **k**.

1J. Сумма последовательности – 2

В этой задаче нужно, как и в задаче «Футбол» найти сумму последовательности. Только элементы поступают с консоли. Общий вид цикла будет таким:

```
while(...)
{
     // ...чтение t
     // ...увеличение s на t
}
```

Вопрос, как его остановить, ведь количество чисел неизвестно!

Будем хранить два числа «предыдущее число» **p** и «текущее» **t**. Каждый раз в цикле читается новое текущее, а прежнее «текущее» устаревает, предыдущее становится этим предыдущим текущим. Цикл останавливаем тогда, когда оба числа **p** и **t** станут равны нулю:

```
while(p != 0 || t != 0)
{
     s += t;
     p = t;
     t = in.nextInt();
}
```

Осталось правильно проинициализировать **p** и **t**, придать им начальные значения. Проще всего это сделать прямо со ввода. Два числа, хотя бы нули, точно будут!

```
int p = in.nextInt();
int t = in.nextInt();
int s = _____;
while(p != 0 || t != 0)
{
     s += t;
```

```
        p = t;
        t = in.nextInt();
}
```

Важен ли порядок первых двух строчек? А чем проинициализировать s? (Подсказка 1.5)

Заметим, что массив в решении заводить не нужно! То есть, если вы еще не очень хорошо представляете себе, что такое массив, то это даже хорошо.

Обязательно дорешайте эту задачу. Решение обязательно вам пригодится в дальнейшем.

Занятие 1.Справочник

Оформление решения и ввод-вывод

Подробное описание взято с сайта **http:// timus.ru**

Программа на Java, посылаемая на проверку, должна содержать ровно один public класс. Этот класс может называться как угодно и он должен содержать метод

```
public static void main(String[] args)
```

Кроме того, программа может содержать любое количество вложенных классов и глобальных непубличных классов.

Вот пример решения задачи «A+B»:

```
import java.io.*;
import java.util.*;

public class Sum
{
public static void main(String[] args)
{
     Scanner in = new Scanner(System.in);
     PrintWriter out = new PrintWriter(System.out);

     int a = in.nextInt();
     int b = in.nextInt();
     out.println(a + b);

     out.flush();
     in.close();
}
```

Это решение (с несколькими классами) тоже является правильным:

```
import java.util.*;

public class Sum2
{
     class AndSuchClassesToo {}
     public static void main(String[] args)
     {
          Scanner in = new Scanner(System.in);
          System.out.println(in.nextInt() + in.nextInt());
     }
}
```

```
class YouCanUseSuchClasses {}
```

Обратите внимание: публичный класс с функцией main должен быть объявлен первым.

Ввод/вывод в Java может стать очень медленным, если пользоваться им неправильно. Вот несколько правил, соблюдая которые, вы сможете избежать проблем, связанных со вводом/выводом:

`Scanner` является самым удобным средством для чтения входных данных в большинстве задач, но скорость его работы оставляет желать лучшего. Используйте его только для чтения небольших входных данных.

Функции классов PrintWriter и Scanner

Конструкторы PrintWriter

PrintWriter(OutputStream out) Creates a new PrintWriter, without automatic line flushing, from an existing OutputStream.
PrintWriter(String fileName) Creates a new PrintWriter, without automatic line flushing, with the specified file name.

Примеры использования

```
PrintWriter out = new PrintWriter (System.out);
```

или

```
PrintWriter out = new PrintWriter ("out.txt");
```

Функции класса PrintWriter

void	**print**(String s) Prints a string.
void	**println**(String s) Prints a String and then terminates the line.

Конструктор класса Scanner

Scanner(InputStream source)

Constructs a new `Scanner` that produces values scanned from the specified input stream.

Пример использования

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

Функции класса Scanner

String	**next**() Finds and returns the next complete token from this scanner.
BigDecimal	**nextBigDecimal**() Scans the next token of the input as a `BigDecimal`.
BigInteger	**nextBigInteger**() Scans the next token of the input as a `BigInteger`.
BigInteger	**nextBigInteger**(int radix) Scans the next token of the input as a `BigInteger`.
boolean	**nextBoolean**() Scans the next token of the input into a boolean value and returns that value.
byte	**nextByte**() Scans the next token of the input as a `byte`.
byte	**nextByte**(int radix) Scans the next token of the input as a `byte`.
double	**nextDouble**() Scans the next token of the input as a `double`.
float	**nextFloat**() Scans the next token of the input as a `float`.
int	**nextInt**() Scans the next token of the input as an `int`.
int	**nextInt**(int radix) Scans the next token of the input as an `int`.
String	**nextLine**() Advances this scanner past the current line and returns the input that was skipped.

long	**nextLong**() Scans the next token of the input as a long.
long	**nextLong**(int radix) Scans the next token of the input as a long.

BufferedReader обеспечивает достаточно быстрый ввод для большинства задач. Но самостоятельно этот класс позволяет лишь читать отдельные символы и строки. Для чтения токенов и чисел используйте **StringTokenizer** или **StreamTokenizer**.

Подробно использование этих классов описано в Справочнике к занятию «Однопроходные алгоритмы».