Занятие №8. Символы и строки в Java

Работа со строками часто очень похожа на работу с массивами. Строки – это фактически последовательности символов.

Символы

Символы в Java представлены типом char. Это 16-битный тип. В нем хранится код символа в кодировке Unicode.

```
System.out.print("H" + "a");
System.out.print('H' + 'a');
```

Что напечатают эти две строчки кода? (Подсказка 8.1)

Мы не будем говорить подробно о кодировках и о проблеме русских букв - кириллицы. Это известная проблема. Дело в том, что для англоязычных разработчиков национальные алфавиты неактуальны. Обычно библиотеки языка позволяют работать с русскими буквами, но использовать их трудно и часто содержат ошибки. Если с латинским – английским - алфавитом, цифрами, знаками препинания все стандартно, то работа с кириллицей зависит и от языка программирования и от операционной системы. Поэтому **в олимпиадных задачах на строки кириллица практически не используется**. И мы в этом занятии не будем касаться этой важной для практического программирования темы.

Символьные константы записываются в одинарных кавычках (апострофах). Например, чтобы создать символьную переменную `comma` со значением «,» можно написать

```
char comma = ',';
```

Чтобы прочитать символам с консоли или из файла, удобнее всего считать всю строку и обратиться к ее символам. Как это сделать подробно описано в разделе этого урока «Класс String».

Каждый символ имеет код – числовое значение. Латинские буквы и цифры расположены в кодовой таблицы подряд. Таким образом, код

```
char ch = 'a';
ch++;
out.println(ch);
```

выведет букву ‘b’ (без апострофов, разумеется).

Можно узнать код символа:

```
out.println((int)'A' + "; " + ((int) 'a');
```

выведет 65; 97, что, кстати, говорит о том, что заглавные латинские буквы расположены в кодовой таблице раньше, чем строчные.

Но обычно знать код не нужно. Символы можно сравнивать (как их коды):

```
char ch = 'g';
if (ch > 'a')
{
     out.println("YES");
}
```

Выведет строку «YES».

Например, чтобы определить, является ли данный символ цифрой, можно написать сравнение

```
if (ch >= '0' && ch <= '9')// ...
```

Очень частой важной и неочевидно-значимой для человека задачей является преобразование символа-цифру в число и обратно.

Например, код

```
if (1 < '0') out.print("Really?!");
```

Выведет строку "Really?!" – условие сработает. Действительно, ведь код символа '0' равен 48! Можно для преобразования вычитать из символа число 48, но лучше использовать сам символ '0'. Это нагляднее.

Преобразование символа-цифры в число

```
int k = ch - '0';
```

обратно:

```
char ch = (char)(k + '0');
```

Для удобства можно вообще не задумываться о кодах и использовать функции класса Character – обертки для типа char:

Конструктор:

```
Character(char c)
```

Получение значения:

```
char charValue()
```

перевод символа в цифру, radix–основание системы счисления, часто 10:

```
static int digit (char ch, int radix)
```

обратно:

```
char fordigit (int digit, int radix)
```

Является ли символ буквой:

```
boolean isLetter()
```

Перевод символа в верхний регистр:

```
char toUpperCase (char ch)
```

Для быстрой и удобной работы с символом создаем объект класса Character и вызываем функции этого класса. Например:

```
char c1 = '3';
// Вызываем статическую функцию digit
int d = Character.digit(c1, 10);
System.out.println("Square of " + c1 + " is " + d * d);
char c2 = 'a';
Character ch = new Character(c2); // дальше работа с объектом ch
if (Character.isLetter(c2)) out.println(c2 + " is letter");
out.println("Capitalized " + ch.charValue() + " is " +
                    Character.toUpperCase(c2));
```

Программа напечатает

```
Squareof 3 is 9
a is letter
Capitalized a is A
```

Многие другие полезные функции класса Character перечислены в Справочнике.

Строки – это последовательности символов.

Для работы со строками в Java предусмотрены классы String и StringBuffer.

Первый из них не позволяет изменять хранимую строку. **Все операции по «изменению» объекта String приводят к тому, что создается новая, измененная строка.** Таким образом, результат нужно чему-то присваивать. Например:

```
String s = "abc";
s = s.replace('a', 'b');
```

Теперь в строке **s1** , будет лежать "bbc".Но нужно понимать, как именно это происходит. Создается новая измененная строка и присваивается **s**. Старое значение **s** – строка "abc" удаляется. Это работает достаточно медленно.

При использовании класса StringBuffer мы можем написать

```
StringBuffer s = new StringBuffer("abc");
s.replace(0, 1, "def");// с какого по какой, на что менять
```

и в строке **s** сразу будет **defc**.

Изменится сама строка. Это выглядит логичнее, мы не должны делать присваивание.

В то же время функции других классов часто используют именно String, строки-объекты класса String можно «складывать» - эта операция называется «конкатенация» - обычным плюсом, и даже константы – строки в кавычках – это тоже объекты класса String. Например, выражение "abcd".indexOf("bc") вполне корректно. Чему оно равно, кстати? (Подсказка 8.2.) Поэтому мы будем говорить именно о классе String.

Класс String

Прежде чем использовать строку, ее надо создать.

Впрочем, это необязательно. Мы уже упоминали, что строка-константа (текст в двойных кавычках) является полноценным объектом класса String и к ней можно применять любые методы класса.

Создание строки

```
String str = "abc";
```

Также для создания объектов-строк можно применять конструкторы. Наиболее полезен здесь

`String(char[] value, int offset, int count)`

создает строку из массива символов (берутся символы начиная с индекса offset количеством count).

Пример

```
char[] chs = {'a','b', 'c', 'e', 'z', 'k'};
String str = new String(chs, 3, 2);
```

создается строка str со значением «ce».

Чтение строки

Читать строки можно методоми **`next()`** и **`nextLine()`** класса **`Scanner`**. Первая читает, игнорируя пробельные символы, «по словам». Вторая читает строку сразу целиком.

Например, пользователь вводит строку

```
Test   reading   Strings
```

делая по три пробела между словами, и нажимает Enter (или эта строка написана в файле). На этом вводе код

```
String s1 = in.next();
String s2 = in.next()
String s3 = in.nextLine();
out.println(s1 + "\n" + s2 + "\n" + s3);
```

выведет

```
Test
reading
   Strings
```

Обратите внимание на сдвиг третьей строки. Это как раз три пробела после слова «reading».

Длина строки

Функция **`length`** позволяет определить длину строки:

код

```
out.println("abc".length());
```

напечатает 3.

Обратите внимание. В данном случае это именно функция. И в отличие от массивов нужно использовать пустые скобки для ее вызова.

Сравнение строк

Строки, как и любые объекты в Java, нельзя сравнивать при помощи оператора «==»!

Вместо этого следует использовать функцию equals

```
if (s1.equals(s2))… // а вот так правильно!
```

Для большей информации можно воспользоваться функцией

```
int compareTo(String anotherString)
```

Она возвращает 0, если строки равны, число большее нуля, если строка к которой применяется функция *лексикографически* больше аргумента и отрицательное число, если лексикографически больше аргумент.

«**Лексикографически**» **означает позже стоит в словаре**. *Например, «слон» больше «мухи», потому что начинается на ‘с’, которая идет после ‘м’, но «муха» больше «моськи», потому что ‘у’ идет после ‘о’. Так же «сан» меньше «сани», потому что короче. Заметим, что сравнение слов с русскими буквами может не всегда работать некорректно.*

Добавление к строке

Практически все можно приписать к строке обычным сложением.

Например код

```
String str = "2*12=";
str = str + 24;
```

помещает в строку str "2*12=24", фактически преобразуя число типа int (24) в строку из двух символов ("24").

Преобразование различных типов в строку и обратно

Нужно четко понимать, что объект и строковая его запись совсем не одно и то же. Поясним на примере. `1 + 23` получится 24, `"1" + "23"` будет "123". Поэтому периодически нужно преобреозовывать различные типы в строки и обратно.

Самый простой и быстрый способ преобразования переменой **`x`** любого типа в строку -это прибавить ее к пустой строке (например, `"" + x`).

Преобразование из строки в различные типы делается через специальные функции этих классов, обычно конструкторы от String.

Например

```
BigInteger bi = new BigInteger("12345");
```

создает «длинное целое» **`bi`** со значением 12345.

Для преобразования примитивных типов обычно используют статические функции классов-оберток. Например, 1 + Integer.parseInt("23") дает именно 24.

Извлечение символа и подстроки

```
char charAt(int index)

String substring(int beginIndex)
```

```
String substring(int beginIndex, int endIndex)
```

Например, удаление символа с индексом 5 из строки str можно сделать так:

```
str = str.substring(0,4) + str.substring(6);
```

Поиск в строке

Для поиска символа или подстроки в строке удобно использовать функции

```
int indexOf(int ch)

int indexOf(int ch, int fromIndex) .// начиная с позиции fromIndex

int indexOf(String str)

int indexOf(String str, int fromIndex) // начиная с позиции fromIndex
```

Они возвращают индекс первого вхождения или -1, если символ или подстрока не обнаружены.

Соответствующие им функции lastIndexOf возвращают индекс последнего вхождения

Посмотрите примеры

Вызов функции	Результат
`"abracadabra".indexOf('k')`	4
`"abrakadabra".lastIndexOf("ab")`	7
`"abrakadabra".indexOf("br", 3)`	8 (первое вхождение подстроки «br», начиная со второго символа 'a')
`"abrakadabra".indexOf("c")`	-1

Функции замены

Группа функций replace позволяют быстро просто осуществлять замены в строке.

```
String replace(char oldChar, char newChar)

String replaceAll(String regex, String replacement)

String replaceFirst(String regex, String replacement)
```

Еще раз обратим внимание. Эти функции не производят изменения строки. Для «эффекта» изменения необходимо использовать присваивание. Например, код

```
String s = "abacabcc"
s = s.replaceAll("ab","def");
out.println(s);
```

ведет `defacdefcc`

Важной в практической работе со строками является функция trim:

`String trim()`

Она возвращает строку без начальных и конечных пробелов.

Для преобразование регистра в классе String служат функции

`String toUpperCase()`

`String toLowerCase()`

Очень мощным средством при работе со строками является функция split. Она разбивает строку в массив строк, используя указанную строку как разделитель.

String s = `"boo:and:foo"`

Вызов функции	Результат – массив
`s.split(":")`	`{"boo", "and", "foo"}`
`s.split("o")`	`{"b", "", ":and:f"}`
`s.split("oo")`	`{"b", ":and:f"}`
`s.split("x")`	`{"boo:and:foo"`

Этой функцией часто пользуются для разделения строки на слова. Например, в задаче «Количество слов» нужно посчитать количество слов. Для этого можно посчитать количество пробелов и прибавить 1. Но в классе String нет специальной функции count. Можно поступить так:

```
out.println(s.split(" ").length());
```

В одной строке разбиваем строку в массив и выводим его длину. Конечно, это «из пушки по воробьям», но работает!

В заключение о перевороте строки.

Разворот строки

В олимпиадных задачах часто нужно перевернуть строку или ее часть, заменить порядок следования символов на обратный. В классе String нет специальной функции reverse, но она есть в классе StringBuffer. Можно воспользоваться ей.

Этот код «перевернет» всю строку

```
s = (new StringBuffer(s)).reverse();
```

а этот - пять начальных символов строки **s**

```
s = (new StringBuffer(s.substring(0,5))).reverse() +
s.substring(6);
```

Немного о задачах

Задач на этом занятии предлагается больше чем обычно – пятнадцать. Это потому, что многие из них решаются быстро средствами самого языка. Однако, некоторые из них надо решать, рассматривая строку посимвольно.

Обратите особое внимание на задачу «Капитан Флинт» в ней технически необычное условие. Читать входные данные в ней надо до конца файла:

```
while(in.hasNext())
{
      …
```

Задачи

Задача A. Проверить, является ли символ цифрой

Определите, является ли данный символ цифрой или нет.

Формат входных данных

Задан единственный символ **c**.

Формат выходных данных

Необходимо вывести строку `yes`, если символ является цифрой, и строку `no` в противном случае.

Пример

Ввод	**Вывод**
a	no

Задача B. Изменить регистр символа

Измените *регистр символа, если он был латинской буквой: сделайте его заглавным*, если он был строчной буквой и наоборот. Для этого напишите отдельную функцию, меняющую регистр символа.

Формат входных данных

Задан единственный символ **C.**

Формат выходных данных

Необходимо вывести получившийся символ.

Задача C. Количество слов

Дана строка, содержащая пробелы. Найдите, сколько в ней слов (слово – это последовательность непробельных символов, слова разделены одним пробелом, первый и последний символ строки – не пробел).

Формат входных данных

На вход подается несколько строк.

Формат выходных данных

Необходимо вывести количество слов в первой из введенных строк.

Пример

Ввод	**Вывод**
`ba 12 ak1` `af`	`3`

Задача D. Самое длинное слово

Дана строка, содержащая пробелы. Найдите в ней самое длинное слово, выведите это слово и его длину.

Формат входных данных

Задана одна строка, содержащая пробелы. Слова разделены ровно одним пробелом.

Формат выходных данных

Необходимо вывести самое длинное слово в строке и его длину.

Пример

Ввод	**Вывод**
`a aa aaa aaaa`	`aaaa` `4`

Задача E. Является ли слово палиндромом?

По данной строке определите, является ли она палиндромом (то есть, можно ли прочесть ее наоборот, как, например, слово "`топот`").

Формат входных данных

На вход подается 1 строка без пробелов.

Формат выходных данных

Необходимо вывести `yes`, если строка является палиндромом, и `no` в противном случае.

Пример

Ввод	**Вывод**
`abba`	`yes`

Задача F. Две одинаковые буквы

Дана строка. Известно, что она содержит ровно две одинаковые буквы. Найдите эти буквы.

Формат входных данных

На вход подается 1 строка.

Формат выходных данных

Необходимо вывести букву, которая встречается в строке дважды.

Пример

Ввод	**Вывод**
`avdaf`	`a`

Задача G. Поиск подстроки

Даны две строки. Определите, является ли первая строка подстрокой второй строки.

Формат входных данных

На вход подается 2 строки.

Формат выходных данных

Необходимо вывести слово `yes`, если первая строка является подстрокой второй строки, или слово `no` в противном случае.

Пример

Ввод	**Вывод**
`ya` `nya`	`yes`

Задача H. Капитан Флинт

Капитан Флинт зарыл клад на Острове сокровищ. Он оставил описание, как найти клад. Описание состоит из строк вида: `"North 5"`, где слово – одно из `"North"`, `"South"`,

"East", "West", – задает направление движения, а число – количество шагов, которое необходимо пройти в этом направлении.

Напишите программу, которая по описанию пути к кладу определяет точные координаты клада, считая, что начало координат находится в начале пути, ось OX направлена на восток, ось OY – на север.

Формат входных данных

На вход подается последовательность строк указанного формата. Гарантируется, что числа не превосходят 108.

Формат выходных данных

Необходимо вывести координаты клада – два целых числа через пробел. Гарантируется, что эти числа не превосходят 108.

Пример

Ввод	**Вывод**
North 5 East 3 South 1	3 4

Задача I. Является ли строка палиндромом?

Дана строка, состоящая из строчных латинских букв и пробелов. Проверьте, является ли она палиндромом без учета пробелов (например, "аргентина манит негра").

Формат входных данных

На вход подается 1 строка, содержащая пробелы. Подряд может идти произвольное число пробелов.

Формат выходных данных

Необходимо вывести yes, если данная строка является палиндромом, и no в противном случае.

Пример

Ввод	**Вывод**
abb a	yes

Задача J. Удали пробелы

Дана строка, Вам требуется преобразовать все идущие подряд пробелы в один.

Формат входных данных

Длина строки не превосходит 1000.

Формат выходных данных

Выведите измененную строку.

Пример входных и выходных данных

Входные данные	Выходные данные
␣d␣hky␣k	␣d␣hky␣k
␣␣tnbnv␣␣l	␣tnbnv␣l

Задача K. Негласный палиндром

Источник: Турнир Архимеда 2007 года

Возьмем произвольное слово и проделаем с ним следующую операцию: поменяем местами его первую согласную букву с последней согласной буквой, вторую согласную букву с предпоследней согласной буквой и т.д. Если после этой операции мы вновь получим исходное слово, то будем называть такое слово *негласным палиндромом*. Например, слова sos, rare, rotor, gong, karaoke являются негласными палиндромами.

Вам требуется написать программу, которая по данному слову определяет, является ли оно негласным палиндромом.

Входные данные

Вводится одно слово.

Выходные данные

Программа должна вывести `YES`, если введенное слово является негласным палиндромом, и `NO` в противном случае.

Примеры

Входные данные	Выходные данные	Комментарий
`tennete`	`YES`	Первая согласная буква – `t` – меняется местами с последней, тоже `t`, вторая согласная буква – `n` – меняется местами с предпоследней, тоже `n`.
`karaoke`	`YES`	Первая согласная буква – `k` – меняется местами с последней, тоже `k`, вторая согласная буква является одновременно и

		предпоследней, поэтому остается на месте.
`disk`	`NO`	Это слово не является негласным палиндромом, поскольку при указанных перестановках букв получается слово `kisd`.

Ограничения

Во всех тестовых примерах слово записано строчными латинскими буквами и состоит не более, чем из 20 символов.

Задача L. Пароль

Источник: Московская командная олимпиада для 8 классов 2007 года

Пароль называется *криптостойким*, если он включает в себя и строчные латинские буквы, и заглавные латинские буквы, и цифры, при этом его длина должна быть не менее 8 символов.

Требуется по данному паролю определить, является ли он криптостойким.

Входные данные

Вводится одна строка, состоящая только из латинских букв и цифр. Количество символов в строке не превышает 100.

Выходные данные

Выведите слово `YES`, если указанный пароль является криптостойким, и `NO` – в противном случае (заглавными латинскими буквами).

Примеры

Входные данные	Выходные данные
`1aA`	`NO`
`AaBbCc12`	`YES`
`AAAaaaAAA`	`NO`

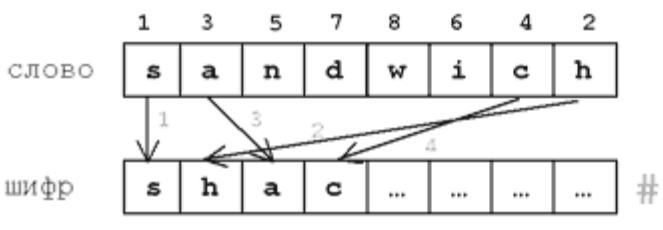
Задача M. Метод бутерброда

Источник: Московская олимпиада, окружной этап 2007 года

Секретное агентство «Super-Secret-no» решило для шифрования переписки своих сотрудников использовать «метод бутерброда». Сначала буквы слова нумеруются в таком порядке: первая буква получает номер 1, последняя буква - номер 2, вторая – номер 3, предпоследняя – номер 4, потом третья … и так для всех букв (см. рисунок). Затем все

буквы записываются в шифр в порядке своих номеров. В конец зашифрованного слова добавляется знак «диез» (#), который нельзя использовать в сообщениях.

Например, слово «sandwich» зашифруется в «shacnidw#».



На рисунке показано начало шифрования слова «sandwich»

К сожалению, программист «Super-Secret-no», написал только программу шифрования и уволился. И теперь агенты не могут понять, что же они написали друг другу. Помогите им.

Формат входных ***данных***

Вводится слово, зашифрованное методом бутерброда. Длина слова не превышает 20 букв.

Формат выходных данных

Выведите расшифрованное слово.

Пример

Входные данные	Выходные данные
Aabrrbaacda#	Abracadabra

Задача N. Хорошие стихи

Источник: Турнир Архимеда 2010

Вы когда-нибудь задумывались над тем, как отличить хорошие стихи от посредственных?

Нет? А вот редактор литературного журнала занимается этим каждый день, получая тонны корреспонденции от молодых авторов, желающих стать известными поэтами. Благо, в последнее время большая часть стихов присылается по электронной почте, поэтому у редактора возникла мысль автоматизировать процесс. Он твердо уверен, что стихи тем лучше, чем точнее в них рифма. Он считает две строки зарифмованными, если у них совпадает несколько последних букв. И чем больше букв совпадает, тем лучше зарифмованы строки. Например, у строк “палка” и “веревка” совпадают только пары последних букв “ка”, а у строк “олимпиада” и “рая и ада” совпадают четыре буквы (пробелы мы пропускаем). Поэтому вторая рифма лучше. Редактор считает, что в четверостишии (четыре строки) первая строка должна рифмоваться с третьей, а вторая – с четвертой. Для каждой из этих двух пар строк он считает количество совпадающих последних символов и из этих двух чисел выбирает наибольшее. Полученное число он называет коэффициентом качества стихотворения – чем он выше, тем больше шансов у

стихотворения быть опубликованным. Помогите редактору – напишите программу, которая определяет качество стихотворения. И кто знает, может быть, благодаря вашим усилиям, мир познакомится с гениальными стихами (см. первый пример).

Формат входных данных

На вход подается 4 непустые строки, каждая из которых состоит из не более чем 100 строчных латинских букв (стихотворение уже подверглось предварительной обработке: из него удалили все пробелы и знаки препинания, а заглавные буквы сделали строчными).

Формат выходных данных

Выведите одно число – коэффициент качества стихотворения.

Примеры

Входные данные	Выходные данные
yapomnyuchudnoemgnovenje peredomnojyavilasty kakmimoletnoevidenje kakgenijchistoykrasoty	4
eto vovse ne stihi	0
etootlichnyestihi etootlichnyestihi etootlichnyestihi etootlichnyestihi	17

Задача O. Благозвучное слово

Источник: Московская окружная олимпиада, 2008 год

Все буквы латинского алфавита делятся на гласные и согласные. Гласными буквами являются: `a, e, i, o, u, y`. Остальные буквы являются согласными.

Слово называется благозвучным, если в этом слове не встречается больше двух согласных букв подряд и не встречается больше двух гласных букв подряд. Например, слова `abba, mama, program` — благозвучные, а слова `aaa, school, search` — неблагозвучные.

Вводится слово. Если это слово является неблагозвучным, то разрешается добавлять в любые места этого слова любые буквы. Определите, какое минимальное количество букв можно добавить в это слово, чтобы оно стало благозвучным.

Входные данные

Вводится слово, состоящее только из маленьких латинских букв. Длина слова не превышает 30 символов.

Выходные данные

Выведите минимальное число букв, которые нужно добавить в это слово, чтобы оно стало благозвучным.

Примеры

Пример ввода	Пример вывода	Комментарий
`program`	`0`	Слово уже является благозвучным
`school`	`1`	Достаточно добавить одну гласную букву, например, между буквами s и c

Подсказки и решения. 8 Занятие.

8.1

Будет напечатано… На169! Почему? В первой строке складываются именно строки. Это единственный перегруженный в Java оператор для объектов – результат «На». А вот во втором складываются символы - тип char! По правилам приведения типов они при сложении конвертируются в int как коды символов! Пример взят из занимательной книги по языковым сложностям языка Java **“Java™ Puzzlers: Traps, Pitfalls, and Corner Cases”.**

8.2

Проверка знания, чувства английского языка. Программистам знание английского просто необходимо! Функция indexOf возвращает индекс первого вхождения подстроки в строку, в нашем случае результат будет 2. Впрочем, и без знания английского можно догадаться!

Разбор. Занятие 8

8A Проверить, является ли символ цифрой

Условие проверки разбирается в тексте занятия. Опишем решение подробно.

Читаем строку и берем из нее нулевой символ. Это можно сделать одной строкой

```
char c = in.next().chatAt(0);
```

И пишем условие

```
if(c >= '0' || c <= '9') out.println("yes");
else out.println("no");
```

Удобнее воспользоваться статической функцией класса Character.

```
if(Character.isDigit(c)) out.println("yes");
else out.println("no");
```

Заметим, что об действия, чтение и проверку, можно сделать одной командой

```
if(Character.isDigit(in.next().charAt(0))) out.println("yes");
else out.println("no");
```

хотя код при этом явно теряет в наглядности.

8B. Изменить регистр символа

Если символ является заглавной буквой – преобразуем его к строчной. Иначе – если строчной, к заглавной. (Важно делать это именно с else, потому что иначе заглавная буква преобразуется в строчную, а потом произойдет обратное преобразование). Для всех действий удобно использовать статические функции класса Character, они описаны в Справочнике:

```
char ch = in.next().charAt(0); // читаем символ
if (Character.isUpperCase(ch)) ch = Character.toLowerCase(ch);
else if (Character.isLowerCase(ch)) ch =
                                    Character.toUpperCase(ch);
out.println(ch);
```

В принципе, вторую проверку можно не делать. Функция toUpperCase сработает корректно даже если символ не будет строчной буквой. В этом случае символ не изменится.

Осталось вывести результат:

```
out.println(c);
```

Задача 8C. Количество слов

Разбирается в тексте занятия.

Задача 8D.Самое длинное слово.

Задача достаточно сложная, если решать ее «посимвольно». Но в Java она решается просто. Разобьем строку функцией split по пробелам. Получим массив слов:

```
String[] word = s.split(" ");
```

Дальше пройдем по нему и найдем номер слова с максимальной длиной:

```
int maxlen = word[0].length();
int imax = 0; // imax – индекс максимального
for (int i = 0; i < word.length, i++)
{
     if (word[i].length() > maxlen)
     {
          maxlen = word[i].length();
          imax = i;
}
```

после цикла выводим word[imax].

8E. Является ли слово палиндромом?

Проще всего создать копию строки,

```
String s2 = new String(s);
```

перевернуть ее при помощи StringBuffer, как описано в тексте занятия, и сравнить. Не забывайте, что строки в Java следует сравнивать функцией **`equals`**, а не оператором «**`==`**»!

8F. Две одинаковые буквы

```
out.print(s.findTwoSameLetters());
```

Все!..

Не компилируется? Жаль…

В этой задаче нам придется действовать посимвольно. Для каждого символа попробуем найти пару. Подобная задача разбиралась в теме «Массивы». Здесь решение абсолютно аналогично.

```
for(int i = 0; i < s.length(); i++)
{
     char c1 = s.charAt(i);
     for(int j = i + 1; j<s.length(); j++)
     {
          c2 = s.charAt(j);
          if (c1 == c2) …// запоминаем найденный символ
     }
}
```

Заметим, что лучше, после того как нашли два одинаковых символа, выйти из циклов. Но это необязательно. В случае, если искомые буквы - последние, циклы все равно будут «крутиться» до конца.

8G. Поиск подстроки

Пример того, как вызывать функцию indexOf, можно найти в тексте занятия. Она решит все проблемы!

8H. Капитан Флинт

Единственная сложность в задаче во время закончить ввод. Ведь количество команд не дано! Делается это на самом деле просто.

```
while(in.hasNext())
```

Каждую строку разбиваем функцией split на команду и аргумент и «выполняем»:

```
String command = in.nextString();
String[] arr = command.split(" ");
if (arr[0].equals("East"))
{
 // Восток – уменьшаем x
 x -= Integer.parseInt(arr[1]);//надо перевести расстояние в int
}
// ...для других сторон света аналогично
```

8I. Является ли строка палиндромом?

Удаляем из строки пробелы

```
s = s.replace(" ", "");
```

и проверяем на палиндром как в задаче 8E.

8J. Удали пробелы

Можно заменять два пробела на один, пока подстрока из двух пробелов находится в строке.

8K Негласный палиндром

Удалим из строки все гласные (посмотрите по словарю). Если полученная строка палиндром, то и первоначальная тоже.

8L. Пароль

Заведем три флажка созначениями false по типам криптостойкости.

```
boolean foundUperCase = false;
boolean foundLowerCase = false;
boolean foundDigit = false;
```

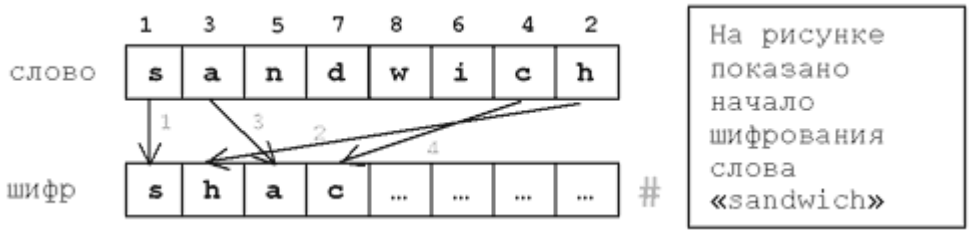
Про каждый символ строки выясним, к какому типу он относится, и поменяем значение соответствующему флажку. Проверять тип удобнее всего функциями класса Character. Например,

```
if (s[i].isUpperCase()) foundUpperCase = true;
```

Если все три флажка оказываются истинными – выводим «YES», иначе “NO”.

8M Метод бутерброда

Приведем для наглядности картинку из у словия.



Заметим, что буквы с нечетных мест шифра в основной строке должны идти последовательно, а с четных в обратном порядке.

Заведем две строки для левой и правой части

```
String sleft = "", sright = "";
```

и будем добавлять к ним буквы слева и справа соответственно:

```
inti = 0;
while (s.charAt(i) != '#') // по условию шифр заканчивается
диезом
{
     if (i % 2 == 0) // в задаче места нумеруются с единицы
     {
          sleft = sleft + s.charAt(i);
}
else
{
```

```
     sright =s.charAt(i) + sright; // приписываем слева
}
}
```

В конце сложим их.

```
out.println(sleft + sright);
```

Это и будет ответ.

8M. Хорошие стихи

Для каждой пары нужных строк, пробегаем циклом по символам с конца до тех пор пока встретим несовпадение или короткая строка не кончится и увеличиваем счетчик:

```
int i1 = s1.length() - 1;
int i3 = s3.length() - 1;
int good13 = 0; // счетчик "хорошести" первой и третьей строк
for (;i1 >= 0 && i3 >= 0 && s1.charAt()==s3.charAt(); i1--,i3--)
{
     good13++;
}
```

Для второй и четвертой строки делаем аналогично.

В конце выводим наибольший из счетчиков(good13и good24).

8O. Благозвучное слово

Понятно, что для того, чтобы сделать слово благозвучным, нужно разбивать подряд идущие символы одного типа (гласные и согласные) символами другого типа.

Для этого будем идти циклом по строке и, если тип текущего символа не отличается от типа предыдущего, увеличить на 1 переменную `k`, в противном случае прибавить к текущему ответу `k/2-1` и установить значение `k` равным единице.

Как определить что буква гласная? Проще всего завести строку из гласных и проверять наличие символа в ней функцией indexOf.

Занятие 8.Справочник

Полезные функции класса Character

char	**charValue**() Returns the value of this Character object.
int	**compareTo**(Character anotherCharacter) Compares two Character objects numerically.
static int	**digit**(char ch, int radix) Returns the numeric value of the character ch in the specified radix.
static char	**forDigit**(int digit, int radix) Determines the character representation for a specific digit in the specified radix.
static boolean	**isDigit**(char ch) Determines if the specified character is a digit.
static boolean	**isHighSurrogate**(char ch) Determines if the given char value is a high-surrogate code unit (also known as *leading-surrogate code unit*).
static boolean	**isLetter**(char ch) Determines if the specified character is a letter.
static boolean	**isLetter**(int codePoint) Determines if the specified character (Unicode code point) is a letter.
static boolean	**isLetterOrDigit**(char ch) Determines if the specified character is a letter or digit.
static boolean	**isLetterOrDigit**(int codePoint) Determines if the specified character (Unicode code point) is a letter or digit.
static boolean	**isLowerCase**(char ch) Determines if the specified character is a lowercase character.
static boolean	**isLowerCase**(int codePoint) Determines if the specified character (Unicode code point) is a lowercase character.

static boolean	**isUpperCase**(char ch) Determines if the specified character is an uppercase character.
static boolean	**isUpperCase**(int codePoint) Determines if the specified character (Unicode code point) is an uppercase character.
static boolean	**isWhitespace**(char ch) Determines if the specified character is white space according to Java.
static char	**toLowerCase**(char ch) Converts the character argument to lowercase using case mapping information from the UnicodeData file.
String	**toString**() Returns a String object representing this Character's value.
static String	**toString**(char c) Returns a String object representing the specified char.
static char	**toUpperCase**(char ch) Converts the character argument to uppercase using case mapping information from the UnicodeData file.

Полезные функции класса String

char	**charAt**(int index) Returns the char value at the specified index.
int	**compareTo**(String anotherString) Compares two strings lexicographically.
int	**compareToIgnoreCase**(String str) Compares two strings lexicographically, ignoring case differences.
boolean	**endsWith**(String suffix) Tests if this string ends with the specified suffix.
byte[]	**getBytes**() Encodes this String into a sequence of bytes using the platform's default charset, storing the result into a new byte array.
int	**indexOf**(int ch)

	Returns the index within this string of the first occurrence of the specified character.
int	**indexOf**(int ch, int fromIndex) Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index.
int	**indexOf**(String str) Returns the index within this string of the first occurrence of the specified substring.
int	**indexOf**(String str, int fromIndex) Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.
int	**lastIndexOf**(int ch) Returns the index within this string of the last occurrence of the specified character.
int	**lastIndexOf**(int ch, int fromIndex) Returns the index within this string of the last occurrence of the specified character, searching backward starting at the specified index.
int	**lastIndexOf**(String str) Returns the index within this string of the rightmost occurrence of the specified substring.
int	**lastIndexOf**(String str, int fromIndex) Returns the index within this string of the last occurrence of the specified substring, searching backward starting at the specified index.
int	**length**() Returns the length of this string.
String	**replace**(char oldChar, char newChar) Returns a new string resulting from replacing all occurrences of oldChar in this string with newChar.
String	**replace**(CharSequence target, CharSequence replacement) Replaces each substring of this string that matches the literal target sequence with the specified literal replacement sequence.
String	**replaceAll**(String regex, String replacement) Replaces each substring of this string that matches the given regular expression with the given replacement.

String	**replaceFirst**(String regex, String replacement) Replaces the first substring of this string that matches the given regular expression with the given replacement.
String[]	**split**(String regex) Splits this string around matches of the given regular expression.
boolean	**startsWith**(String prefix) Tests if this string starts with the specified prefix.
boolean	**startsWith**(String prefix, int toffset) Tests if the substring of this string beginning at the specified index starts with the specified prefix.
String	**substring**(int beginIndex) Returns a new string that is a substring of this string.
String	**substring**(int beginIndex, int endIndex) Returns a new string that is a substring of this string.
char[]	**toCharArray**() Converts this string to a new character array.
String	**toLowerCase**() Converts all of the characters in this String to lower case using the rules of the default locale.
String	**toUpperCase**() Converts all of the characters in this String to upper case using the rules of the default locale.
String	**trim**() Returns a copy of the string, with leading and trailing whitespace omitted.
static String	**valueOf**(boolean b) Returns the string representation of the boolean argument.
static String	**valueOf**(char c) Returns the string representation of the char argument.
static String	**valueOf**(double d) Returns the string representation of the double argument.
static String	**valueOf**(float f) Returns the string representation of the float argument.

static String	**valueOf**(int i) Returns the string representation of the int argument.
static String	**valueOf**(long l) Returns the string representation of the long argument.