**2. Статистика символов.**

Создайте класс, который хранит строку во внутреннем поле, со следующим интерфейсом:

**class Message {**

public Message(String str); //конструктор содержит код, который выполняется при создании объекта. Это код, который запускается, когда вы пишете слово new перед типом класса.

public Message(); //еще один конструктор, позволяющий создать пустую строку.

public int countWords(); // возвращает количество слов

public String reverse(); // возвращает перевернутую строку

public int count(char c); // возвращает кол-во вхождений символа в строку

public String nomer(); // возвращает номер (номер находится после №, но после № может пробела и не быть). Пример. Строка Паспорт СеРия 6655 №1111 выдан, возвращает 1111; строка Паспорт сеРия АА № 2345, возвращает 2345.

public String replaceSubstring(String sold, String snew); // возвращает строку, в которой определенное слово заменено другим.

**}**

В отдельном классе Test опишите метод main(...), который считывает строку из консоли и выводит результат каждого метода класса.

**Указания к работе.**

Нужны следующие члены класса:

1. Поле String Stroca;

2. Конструктор, позволяющий создать строку на основе заданной строки

public Message(String str)//конструктор

{

Stroca = str;

}

3. Конструктор, позволяющий создать пустую строку

public Message()//конструктор

{

…;//написать самим

}

4. Метод, который возвращает перевернутую строку

public String reverse(){

String rev="";

int length= Stroca.length();

for(int i=length-1;i>=0;i--)

rev=rev+ Stroca.charAt(i);

return rev;

}

Остальные методы дописываем.

**Ввод строки.**

1. Для ввода данных используется класс Scanner из библиотеки пакетов Java.

Этот класс надо импортировать в той программе, где он будет использоваться. Это делается до начала открытого класса в коде программы.

Для работы с потоком ввода необходимо создать объект класса Scanner, при создании указав, с каким потоком ввода он будет связан. Стандартный поток ввода (клавиатура) в Java представлен объектом — System.in. А стандартный поток вывода (дисплей) —объектом System.out.

Имеется метод nextLine(), позволяющий считывать целую последовательность символов, т.е. строку, а, значит, полученное через этот метод значение нужно сохранять в объекте класса String. В следующем примере создаётся два таких объекта, потом в них поочерёдно записывается ввод пользователя, а далее на экран выводится одна строка, полученная объединением введённых последовательностей символов.

import java.util.Scanner;  
public class Main {  
**public** **static** **void** main(String[] args) {

Scanner sc = **new** Scanner(System.*in*);

String s1, s2;

s1 = sc.nextLine();

s2 = sc.nextLine();

System.*out*.println(s1 + s2);

}

}

1. Для ввода данных используется диалоговое окно.

Неотъемлемой частью большинства программ являются небольшие диалоговые окна: для вывода пользователю сообщения (например, сообщения об ошибке), для вопроса, ответ на который важен для выполнения текущего действия (например, просьба подтвердить или отменить запрашиваемую операцию). Эти диалоги могут быть запрограммированы вручную на основе класса JFrame. Однако ввиду того, что они являются типичными для многих программ, Swing предоставляет в распоряжение программиста несколько готовых классов для работы с ними.

Чаще всего используется класс JOptionPane, содержащий несколько статических методов, отображающих стандартные диалоги.

**import** javax.swing.\*;

**public** **class** Name {

**public** **static** **void** main(String[] args) {

String name;

name=JOptionPane.*showInputDialog*("Введите имя:");

JOptionPane.*showMessageDialog*(**null**, "Приятно позакомиться,\n"+name+"!");

}

}

**В классе String** существует масса полезных методов, которые можно применять к строкам (перед именем метода будем указывать тип того значения, которое он возвращает):

1. int length() — возвращает длину строки (количество символов в ней);
2. boolean isEmpty() — проверяет, пустая ли строка;
3. String replace(a, b) — возвращает строку, где символ a (литерал или переменная типа char) заменён на символ b;
4. String toLowerCase() — возвращает строку, где все символы исходной строки преобразованы к строчным;
5. String toUpperCase() — возвращает строку, где все символы исходной строки преобразованы к прописным;
6. boolean equals(s) — возвращает истину, если строка к которой применён метод, совпадает со строкой s указанной в аргументе метода (с помощью оператора == строки сравнивать нельзя, как и любые другие объекты);
7. int indexOf(ch) — возвращает индекс символа ch в строке (индекс это порядковый номер символа, но нумероваться символы начинают с нуля). Если символ совсем не будет найден, то возвратит -1. Если символ встречается в строке несколько раз, то возвратит индекс его первого вхождения.
8. int lastIndexOf(ch) — аналогичен предыдущему методу, но возвращает индекс последнего вхождения, если символ встретился в строке несколько раз.
9. int indexOf(ch,n) — возвращает индекс символа ch в строке, но начинает проверку с индекса n (индекс это порядковый номер символа, но нумероваться символы начинают с нуля).
10. int [**indexOf**](https://docs.oracle.com/en/java/javase/18/docs/api/java.base/java/lang/String.html#indexOf(java.lang.String))([**String**](https://docs.oracle.com/en/java/javase/18/docs/api/java.base/java/lang/String.html) str) - возвращает индекс в данной строке первого вхождения указанной подстроки.
11. char charAt(n) — возвращает символ, находящегося в строке под индексом n (индекс это порядковый номер символа, но нумероваться символы начинают с нуля).
12. String trim()-возвращает строку, в которой удалены лишние пробелы в начале строки и в конце.
13. String substring(int beginIndex, int endIndex) - возвращает подстроку данной строки, начиная с символа с индексом beginIndex, заканчивая endIndex - 1.
14. String substring(int beginIndex) - возвращает подстроку данной строки, начиная с символа под номером beginIndex и до конца строки.
15. ***boolean contains(CharSequence s)*** — проверяет, содержит ли строка заданную последовательность символов.
16. String[] split(String regex) - разделяет строку на подстроки, используя разделитель, который определяется с помощью регулярного выражения regex.

Посмотрим еще на несколько примеров того, как бы отработал метод split:

|  |  |  |
| --- | --- | --- |
| Строка | Разделитель | Результат работы метода |
| **"I love Java"** | **" "** (символ пробела) | {**"I"**, **"love"**, **"Java"**} |
| **"192.168.0.1:8080"** | **":"** | {**"192.168.0.1"**, **"8080"**} |
| **"Красный, оранжевый, желтый"** | **","** | {**"Красный"**, **" оранжевый"**, **" желтый"**} |
| **"Красный, оранжевый, желтый"** | **", "** | {**"Красный"**, **"оранжевый"**, **"желтый"**} |

Обрати внимание на различия между последними двумя строками в таблице выше. В предпоследней строке разделителем выступает символ запятой, поэтому строка разбилась таким образом, что в некоторых словах есть ведущие пробелы. В последней же строке в качестве разделителя мы использовали символ запятой и символ пробела. Поэтому в результирующем массиве не оказалось строк с ведущими пробелами. Это просто маленькая деталь, в которой демонстрируется, как важно внимательно подбирать правильный разделитель.