Строки

Строки в Java – это объект класса String. Строки не являются массивами, как во многих языках программирования, поэтому со строкой нельзя работать, обращаясь к переменной, как к массиву. Здесь вся работа строится при помощи методов.

Если мы создадим переменную строки без инициализации, то переменная будет равна null. (см. главу про массивы).

```
String s;
```

Обратите внимание, что имя класса пишется с большой буквы. Имя класса не обязано писаться с большой буквы, но такая нотация считается хорошим тоном.

Поэтому обычно, работая со строками, инициализируют её при создании.

```
String s2 = "Hello, World!";
```

Если строка не должна содержать символов, то её инициализируют пустой строкой:

```
String s3 = "";
```

В Java реализованы две очень важные вещи: автоматическое сложение строк(конкатенация) и автоматическое преобразование нестроковых переменных к строкам.

Это позволяет формировать строки простейшими командами:

```
public class Main {

    public static void main(String[] args) {
        String sa = "a";
        String sb = "b";
        String res = "a+b=";
        int a = 10;
        int b = 5;
        String s = sa+"="+a+", "+sb+"="+b+", "+res+(a+b);
        System.out.println(s);
    }
}
```

На консоль будет выведено:

>> a=10, b=5, a+b=15

Длина строки вычисляется при помощи метода .length().

```
String s = "Hello, World!";

int a = s.length();

System.out.println(a);
```

На консоль будет выведено:

» 13

Т.к. любая строка – это объект класса String, то у неё есть множество полезных методов:

.contains(“подстрока”) – возвращает boolean: true, если подстрока содержится в строке и false, если нет.

```
public class Main {

    public static void main(String[] args) {
        String s = "Hello, World!";
        System.out.println(s.contains("llo"));
    }
}
```

На консоль будет выведено:

>> true

.indexOf(“подстрока”) - возвращает индекс, с которого начинается первое вхождение подстроки в исходную строку.

.lastIndexOf(“подстрока”) - возвращает индекс последнего вхождения подстроки в строку.

Вхождение подстроки – это то место строки, где начинается последовательность символов, которая полностью совпадает с последовательностью символов подстроки.

Если указанная подстрока не встречается в исходной строке методы возвращают число -1.

```
public class Main {

    public static void main(String[] args) {
        String s = "Hello, World!";
        System.out.println(s.indexOf("o"));
        System.out.println(s.indexOf("a"));
        System.out.println(s.lastIndexOf("o"));
    }
}
```

На консоль будет выведено:

>>4

>>-1

>>8

.endsWith(«подстрока») - проверяет, заканчивается ли строка подстрокой

.startsWith(«подстрока») - проверяет, начинается ли строка с подстроки

Строки являются особыми объектами. Строки нельзя менять. Такие объекты называется immutable объекты(неизменяемые). Поэтому строки нельзя менять, можно взять часть строки и на её основе создать новую.

Следующие методы создают новую строку.

.substring() - взятие подстроки из исходной строки. Принимает в качестве параметров либо первый и последний индекс строки, из которой надо получить подстроку, либо только первый индекс и формироет подстроку, начиная с него и до последнего элемента.

```
public class Main {
    public static void main(String[] args) {
        String s = "Hello, World";
        String s1 = s.substring(0,4);
        System.out.println(s1);
        String s2 = s.substring(2);
        System.out.println(s2);
    }
}
```

На консоль будет выведено:

>>Hell

>>llo, World

.trim() - возвращает строку без лишних пробелов вначале строки и в конце.

```
public class Main {
   public static void main(String[] args) {
      String s = "   Hello,  World    ";
      String s1 = s.trim();
      System.out.println(s1);
   }
}
```

В консоль будет выведено:

>Hello, World

Обратите внимание, что лишние пробелы внутри строки не удаляются

Т.к. строки — это объекты, то сравнивать при помощи == их нельзя.

Есть специальный метод .equals(), который возвращает true, если строки совпадают и false, если нет.

```
public class Main {
   public static void main(String[] args) {
      String s1 = "abc";
      String s2 = "cde";
      String s3 = "abc";
      System.out.println(s1.equals(s2));
      System.out.println(s1.equals(s3));
   }
}
```

В консоль будет выведено:

>> false

>> true

.replace(что_заменить,на_что_заменить) — заменяет все вхождения первой подстроки на вторую подстроку.

```
import java.util.Locale;
```

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Locale.setDefault(new Locale("en", "US"));
        Scanner sc = new Scanner(System.in);
        String s = "asd sa sd as bsdjf";
        s = s.replace("sd","111");
        System.out.println(s);
    }
}
```

На консоль будет выведено:

>> a111 sa 111 as b111jf

Чтение строки из консоли.

Для этого есть 2 метода: .nextLine() и .next(). Первый считывает все символ до символа переноса строки, второй считывает все символы до символа переноса строки или пробела. Можно сказать, что .next() читает слова.

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String s = sc.nextLine();
        String s2 = sc.next();
        System.out.println(s);
        System.out.println(s2);
    }
}
```

<<Hello, World!

<<Hello, World!

>>Hello, World!

>>Hello,

Первые две строки — это ввод в консоль, вторые две — вывод.

Есть два метода для работы со строкой как с массивом символов:

.charAt() - по индексу выдаёт символ из строки

.toCharArray() - создаёт на основе строки массив

```
public class Main {
    public static void main(String[] args) {
        String s = "Hello, World!";
        System.out.println(s.charAt(1));
        char [] cArr = s.toCharArray();
        for (char c:cArr){
            System.out.print(c+"/");
        }
    }
}
```

На консоль будет выведено:

>>e

>>H/e/l/l/o/,/ /W/o/r/l/d/!/

Обратите внимание, что в этом коде была использована конструкция for...each.

Регулярные выражения.

Бывают задачи, в которых нам надо проверить, соответствует ли строка нашим требованиям. Эти требования задаются при помощи регулярных выражений.

Регулярные выражения — довольно большая тема, поэтому пока что мы разберём только самые азы.

Регулярное выражение — это особым образом оформленная трока. Регулярное выражение требоющее от строки, чтобы она представляла из себя ровно один символ, лежащий в заданном диапазоне, выглядит так «[0-9]» или так: «[abcde]». Т.е. в квадратных скобках указывается либо диапазон значений, либо они выписываются в явном виде. Чтобы строка состояла из минимум одного (можно и больше) символа, надо после квадратных скобок поставить «+».

Например, регулярное выражение, соответствующее строке, состоящей только из букв от f до k. выглядит так: «[f-k]+».

При помощи регулярных выражений над строкой можно выполнять довольно много операций, но мы разберём две самые простые: .matches() - проверить, что строка полностью удовлетворяет регулярному выражению и .find() - что в строке есть хотя бы одна подстрока, удовлетворяющая ему.

Для работы с регулярным выражением нам необходимы классы Pattern и Matcher.

Пока что можно до конца не углубляться в механику работы и пока что просто выучить конструкцию.

```
public class Main {
    public static void main(String[] args) {
        String regStr = "[0-9]";
        String inS="124124214";
        Pattern p = Pattern.compile(regStr);
        Matcher m = p.matcher(inS);
        System.out.println(m.matches());
        System.out.println(m.find());
        String regStr = "[A-Z]+";
        String inS="ASFASFGRGSBKSRBKSBBSRC";
        p = Pattern.compile(regStr);
        m = p.matcher(inS);
        System.out.println(m.matches());
    }
}
```

На консоль будет выведено:

>> false

>> true

>> true

Обратите внимание, что во второй раз мы не создаём новые переменные класса, т. к. они были уже заданны, а просто присваиваем им новые значения.

Последним мы рассмотрим самый полезный метод класса string. Это split(). Этот метод разделяет строку на массив строк по заданному разделителю. Например, разделитель может быть символом пробела (в этом случае строка разобьётся на массив слов) или «/».

```
public class Main {
    public static void main(String[] args) {
        String regStr = "[0-9]";
        String inS="dafasfasf/asfgas/gasg";
```

```
        String [] sArr = inS.split("/");
        for(String s:sArr){
            System.out.println(s);
        }
    }
}
```

На консоль будет выведено:

>>dafasfasf

>>asfgas

>>gasg

А что если подряд будет два разделителя? Тогда некоторые элементы массива будут просто пустыми строками.

```
public class Main {
    public static void main(String[] args) {
        String regStr = "[0-9]";
        String inS="aasd dfnsjfsd   asndj    asdasdas  ";
        String [] sArr = inS.split(" ");
        for(String s:sArr){
            System.out.println(s);
        }
    }
}
```

На консоль будет выведено:

>>aasd

>>dfnsjfsd

>>

>>

>>asndj

>>

>>

>>

>>asdasdas

>>

Вернёмся к методу split(). На самом деле, split в качестве аргумента принимает регулярное выражение. А т. к. часть символов является особыми, то иногда приходится их экранировать. Например, если в качестве разделителя взять «.», то наш код не сработает, т. к. «.» в регулярных выражениях означает любой символ.

```
String s = "192.168.0.1";

String sArr [] = s.split("."); // не сработает
```

Для этого используется специальная команда экранирования: Pattern.quote("строка”);

```
String s = "192.168.0.1";

String sArr [] = s.split(Pattern.quote(".")); // сработает
```

StringBuilder

Т.к. строки — неизменяемые объекты в Java реализован изменяемый аналог строк. Он Называется StringBuilder.

Чтобы на основе строки создать объект класса StringBuilder, надо в качестве аргумента при создании объекта передать необходимую строку:

```
String s = "Hello, World!";

StringBuilder sb = new StringBuilder(s);
```

Чтобы преобразовать объект StringBuffer обратно к строке, необходимо вызвать метод .toString() или прибавить к нашему объекту пустую строку, что по факту является одним и тем же.

```
String outS = sb.toString();
```

У StringBuffer есть много полезных методов. Обратите внимание, что StringBuilder изменяемый объект, что позволяет редактировать строку, не создавая новых объектов. Помимо этого, многие методы возвращают экземпляр объекта, метод которого был вызван. Это сделано специально для последовательного вызова нескольких методов.

Например, имя_переменной.метод1().метод2().метод3(). Это очень распространённая практика в Java.

.reverse() - разворачивает строку.

```
public class Main {
  public static void main(String[] args) {
    String s = "Hello, World!";
    StringBuilder sb = new StringBuilder(s);
    sb.reverse();
    String outS = sb.toString();
    System.out.println(outS);
  }
```

```
}
```

На консоль будет выведено:

>> !dlroW ,olleH

.insert() - вставляет строку в заданное место

```
public class Main {
  public static void main(String[] args) {
    String s = "Hello, World!";
    StringBuilder sb = new StringBuilder(s);
    sb.insert(2,"|ins|");
    String outS = sb.toString();
    System.out.println(outS);
  }
}
```

На консоль будет выведено:

>> He|ins|llo, World!

.replace() - заменяет символы с индекса, введённого первым до индекса, введённого вторым, заданной строкой

```
public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    String s = "Hello, World!";
    StringBuilder sb = new StringBuilder(s);
    sb.replace(2,8,"|rep|");
    String outS = sb.toString();
    System.out.println(outS);
  }
}
```

На консоль будет выведено:

>> He|rep|orld!

.delete() - удаляет из строки символы, с индексами, начиная с первого числа и до второго.

```
public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
```

```
    String s = "Hello, World!";
    StringBuilder sb = new StringBuilder(s);
    sb.delete(4,8);
    String outS = sb.toString();
    System.out.println(outS);
  }
}
```

.replace(начальны_индекс, конечный_индекс,строка) - заменяет часть строки заданного диапазона, хранящейся в StringBuilder на заданную строку. По факту выполняется сначала delete(), потом insert().

```
public class Main {
  public static void main(String[] args) {
    String s = "Hello, World!";
    StringBuilder sb = new StringBuilder(s);
    sb.replace(2,9,"|rep|");
    System.out.println(sb.toString());
  }
}
```

На консоль будет выведено:

>> He|rep|rld!

.append() - добавляет в конец строки заданную строку или символ.

У класса StringBuilder есть двойник — StringBuffer. Он имеет те же методы, что и StringBuilder, но *синхронизирован*.

Пока что мы не будем подробно разбирать синхронизированные объекты, но нужно знать, что есть два класса StringBuilder и StringBuffer. StringBuilder быстрее, но его нельзя использовать в многопоточных приложениях, в отличие от StringBuffer. StringBuffer работает медленнее.

Последней разберём задачу: как из строки, начинающейся с двузначного числа, взять это число и вывести его, умноженное на 2.

Т.к. коды соседних цифр отличнаются на 1: символ „0” имеет код 48, символ «1» имеет код 49 и т. д., то значение цифры можно получить просто отняв от заданного символа символ «0».

```
public class Main {
  public static void main(String[] args) {
    String s = "29asfasf";
    char c [] = s.substring(0,2).toCharArray();
    int n10 = c[0]-'0'; // число десятков
```

```
    int n1 = c[1]-'0'; // число единиц
    int val = n10*10+n1; // вычисляем число
    System.out.println(val*2);
  }
}
```

На консоль будет выведено:

>> 58