Строки

- В Java поддерживаются 2 *вида строк* :
  - *неизменяемые* – объекты класса **String**;
  - *изменяемые* – объекты класса **StringBuffer**

- Основные свойства «строковых» классов:
  - оба класса принадлежат пакету *java.lang*;
  - оба класса *не могут наследоваться*. Это сделано в целях повышения эффективности работы программ обработки строк

Конструкторы класса String

- Что такое конструктор?

Конструктор - это специальный метод, который вызывается при создании нового объекта.

- Конструктор без параметров создает пустую строку:
  - String s = new String();

- Конструкторы, создающие строки из массива символов:
  - char chars[] = {'a', 'b', 'c', 'd', 'e', 'f'};
  - String s1 = new String(chars); // s1 = "abcdef"
  - String s2 = new String(chars, 2, 3); // s2 = "cde"

Строковые константы

- Строковые константы записываются в виде последовательности символов, заключенных в двойные кавычки

- Каждая такая константа представляет собой полноценный объект класса String, можно вызывать методы этого объекта.
  - String s = "abc";
  - int n = "abc".length();

Ввод строки с клавиатуры

- Для чтения данных из консоли используется стандартный поток ввода System.in
- Чтение данных осуществляется посредством вызова метода readLine();

```
Scanner sc = new Scanner(System.in);
        String s1;
    String s2;
        s1 = sc.nextLine();
        s2 = sc.nextLine();
        System.out.println(s1);
System.out.println(s2);
```

Преобразование строк

- Когда какое-то значение должно быть преобразовано в строку, вызывается статический метод **String.valueOf(…)**.
- Этот метод перегружен ***для всех простых типов***.
  - String pi = String.valueOf(3.14159);
  - String cond = String.valueOf(true);
- А также для **Object**. В этом случае вызывается метод **toString()** объекта.
  - String.valueOf(someObj) // Эквивалентно someObj.toString()

Извлечение символов

- **char charAt (int index)** - возвращает символ строки, стоящий в позиции index

- **void getChars(int sourceStart, int sourceEnd, char target[], int targetStart)** - существующий массив target заполняется частью строки, начиная с позиции **sourceStart** включительно и кончая позицией **sourceEnd** исключительно. Параметр **targetStart** задает место в массиве, с которого начнется его заполнение.

- **char[] toCharArray** создается массив символов и заполняется символами строки.

Извлечение байт

- byte[] **getBytes**() - создается массив байт и заполняется символами строки. Символы предварительно преобразуются в байты, поэтому количество байт массива будет равно количеству символов строки.

- **byte[] getBytes(String encoding)** - то же, но с изменением кодировки

Методы класса String

- int **length**() - возвращает длину строки (количество символов в строке).
- String **concat**(String s) - присоединяет строку s к строке **this**
- String **trim**() - удаляет ведущие и завершающие пробельные символы из строки this (пробельными символами считаются: ' ', \r, \n, \t).
- String **replace**(char original, char replacement) - заменяет все вхождения символа **original** символом **replacement**.
- String **toLowerCase**() - изменяет регистр символов в строке, делая все буквы строчными.
- String **toUpperCase**() - изменяет регистр символов в строке, делая все буквы прописными
- String **replaceAll**(String regex, String replacement)) – замещает все вхождения regex на replacement. В качестве regex может быть регулярное выражение
- String **replaceFirst**(String regex, String replacement) – то же, только замещает первое вхождение

Конкатенация строк

- Для склеивания строк в Java используется оператор “+”
  - String s1 = "lang" + "uage" ; // s1 = language
- Если строка соединяется не со строковым значением простого или ссылочного типа, то последнее преобразуется в строку
  - String s2 = 3 + " pigs" ; // s2 = "3 pigs”
  - String s3 = "four = " + 2 + 2; // s3 = " four = 22 “
  - String s4 = "four = " + (2 + 2); // s4 = “four = 4”
- Добавить к одной строке другую можно с помощью оператора “+=“
  - String gruss = “Hallo”;
  - gruss += “, Freunde”; // gruss будет иметь значение “Hallo, Freunde”;
- Однако следует понимать, что при конкатенации строк создается совершенно новый объект, и значения склеиваемых строк в него копируются.

```
for(int i = 0; i < 10000; i++){
        someString += someArray[i]; // Никогда не делайте так !!!
}
```

Сравнение строк

- boolean **equals**(Object other) – производит посимвольное сравнение строки this со строкой other с учетом регистра символов
- boolean **equalsIgnoreCase**(String other) – то же, но без учета регистра символов
- boolean **regionMatches**(int startIndex, String other, int otherStartIndex, int numChars) - сравнивает между собой два участка строк this и other.
- boolean **regionMatches** (boolean ignoreCase, int startIndex, String str2, int str2StartIndex, int nimChars) – без учета регистра символов

Сравнение строк

- int **compareTo**(String other) - позволяет узнать, какая строка больше и возвращает отрицательное число, если строка this меньше, чем other, ноль, если строки совпадают, и положительное число, если строка this больше, чем строка other.

- boolean **startsWith**(String substr) - проверяет, начинается ли строка this с подстроки substr.

- boolean **endsWith**(String substr) - проверяет, заканчивается ли строка this подстрокой substr.

Поиск в строках

- Задача поиска состоит в том, чтобы определить, в каком месте строки в нее входит символ или другая строка. Это позволяет серия методов **indexOf()**:
  - int **indexOf**(int ch) - возвращает место первого вхождения в строку this символа ch;
  - int **lastIndexOf**(int ch) - возвращает место последнего вхождения в строку this символа ch;
  - int **indexOf** (String substr) - возвращает место первого вхождения в строку this подстроки substr;
  - int **lastIndexOf**(String substr) - возвращает место последнего вхождения в строку **this** подстроки substr;

Модификация строк

- Как известно, изменить объект класса String нельзя
- Можно только создать другую строку, равную части исходной строки
  - String **substring**(int beginIndex) - создает новую строку, равную части строки **this**, начинающейся с позиции **beginIndex**
  - String **substring**(int beginIndex, int endIndex) – создает новую строку, равную участку строки **this**, начиная с позиции **beginIndex** включительно и заканчивая позицией **endIndex** исключительно

Пример работы со строками

- **Пример**. Дано предложение
- "Алгоритмы+Данные=Программы"
- Составить предложение "Алгоритмы=Программы-Данные".

⇓

```
String source = "Алгоритмы+Данные=Программы";
// Определяем позицию символа '+‘
int p1 = source.indexOf('+');
// Определяем позицию символа '=‘
int p2 = source.indexOf('=');
// Вырезаем первое слово
String alg = source.substring(0, p1);
// Вырезаем второе слово
String dat = source.substring(p1 + 1, p2);
// Вырезаем третье слово
String prg = source.substring(p2 + 1);
// Сшиваем все по-новому
String target = alg + "=" + prg + "-" + dat;
```

Изменяемые строки

- Изменяемые строки реализованы в Java с помощью класса **StringBuffer**.
- У этого класса нет многого, из того, что имеет класс String, зато есть несколько версий методов:
  - **append()** - добавления,
  - **delete()** - удаления,
  - **insert()** - вставки подстроки в строку this.

- Кроме того, имеются методы **setCharAt()** и **deleteCharAt()** для замещения и удаления отдельных символов

Пример работы с изменяемыми строками

- **Пример.** Удалить из строки s1 все вхождения подстроки s2

```
String s1 = "qwe12qwe345qwe678qwe90qwe";
String s2 = "qwe";
StringBuffer s = new StringBuffer(s1);
int p = -1;
while ( (p = s.toString().indexOf(s2)) >= 0 ) {
      s.delete(p, p + s2.length());
}
s1 = s.toString();
```