Лекция №9
«Strings»

Автор: доц. каф. ПО ЭВМ ХНУРЭ Колесников Д.О.

9.1. Класс String нельзя расширить

Строки являются объектным типом. Класс String не может быть наследован.

public final class String extends Object
    implements Serializable, Comparable, CharSequence

Доступ к элементам строки можно осуществить с помощью метода charAt(int index)

public char **charAt**(int index)

нумерация символов строки начинается с 0.

Замечание. В отличие от массивов, длина которых содержится в *поле* length, в классе String поле length отсутствует, а длину строки (количество входящих в нее символов) возвращает метод length().

public int length()

9.2. Перегрузка операции «+» для строк

В Java отсутствует перегрузка операций. Исключением является операция сложения «+», которая перегружена для конкатенации (соединения) строк.

При конкатенации (сложении) строк создается новый объект типа String потому что один раз созданный объект типа String нельзя изменить, т.е. нельзя изменить входящие в него символы.

Замечание. Нельзя изменить объект типа String, но можно изменить значение объектной переменной, которая *ссылается* на объект, заставив ее ссылаться на новый объект String.

```
String S = new String("asdf");
S = new String("1234");
S = "ABCD";
```

9.3. Получение строкового представления значений примитивных типов

Для того чтобы получить строковое представление значения примитивного типа следует воспользоваться статическим методом valueOf, который возвращает соответствующую строку.

Данный метод перегружен таким образом, что может принимать значения любых примитивных типов в качестве параметра (кроме void).

Кроме этого метод может принимать в качестве параметра массив символов (char), который будет преобразован в строку.

```
String s = String.valueOf(0xFF); // s = "255"

char[] ch = {'a', 'b', 'c'};
s = String.valueOf(ch); // s = "abc"

s = String.valueOf(-34.4); // s = "-34.4"
```

9.4. Сложение объектных переменных, ссылающихся на null, со строкой

Если строка складывается (конкатенируется) с объектом, который ссылается на null, то вместо объекта подставляется строка "null".

```
String s = null;
s = s + "34"; // s = "null34"
Integer i = null;
s = "abc" + i; // s = "abcnull"
```

Замечание. Если со строкой складывается null, то вместо null подставляется строка "null".

```
String s = "abc" + null; // s = "abcnull"
```

9.5. Строковые литералы (строки-константы)

Строковые литералы (строки-константы) представляют из себя совокупность символов заключенных в двойные кавычки и являются объектными переменными.

Два строковых литерала, состоящих из одного и того же набора символов ссылаются на один и тот же объект-строку.

```
String s1 = "abc";
String s2 = "abc";
boolean flag = (s1 == s2); // flag = true
```

9.6. Пул строк

JVM поддерживает пул строк. В него заносятся все строковые литералы.
Метод intern вызванный на объекте String проверяет, используя метод equals, содержится ли данный объект в пуле. Если содержится, то возвращается ссылка на него. Если не содержится, то объект будет занесен в пул и метод intern вернет на него ссылку.

```
String s = "abc";
String s2 = new String("abc");
boolean f = s == s2; // f = false
f = s == s2.intern(); // f = true
```

```
String s1 = "abc", s2 = "ab";
boolean flag = (s1 == "abc"); // flag = true
flag = ((s1 +"") == "abc"); // flag = false
flag = ((s1 +"").intern() == "abc"); // flag = true

flag = (s1 == "ab" + "c"); // flag = true
flag = (s1 == s2 + "c"); // flag = false
flag = (s1 == (s2 + "c").intern()); // flag = false
```

9.7. Сравнение строк посимвольно

Для сравнения строк посимвольно следует использовать метод equals объекта String.

public boolean equals(Object object)

Для сравнения строк без учета регистра символов, из которых состоит строка, следует использовать метод equalsIgnoreCase класса String.

public boolean equalsIgnoreCase(String anotherString)

Метод **equals** возвращает значение *true* если строка, на которой вызывается метод содержит в точности такую же последовательность символов, как и строка, которая передается методу в качестве аргумента. В противном случае метод вернет значение *false*.

```
boolean flag;
String s1 = "abc", s2 = "ab";
flag = s1.equals("ab" + "c"); // flag = true
flag = s1.equals(s2 + "c"); // flag = true
flag = s1.equals(new String("a") + "bc"); // flag = true
```

Замечание. Метод equals содержит каждый класс, т.к. он определен в классе Object. В классе Object метод определен таким образом, что x.equals(y) эквивалентно (с точки зрения значения результата) x == y.

9.8. Метод substring

Класс String содержит перегруженный метод **substring**, которой возвращает подстроку, определяемую значениями численных индексов начала и конца подстроки, передаваемых в метод в качестве параметров.

// возвращает часть строки, начиная с beginIndex:
public String substring(int beginIndex)

// возвращает часть строки от beginIndex до endIndex-1:
public String substring(int beginIndex, int endIndex)

Замечание. Метод substring выбрасывает исключение **java.lang.IndexOutOfBoundsException** в трех случаях:

1) beginIndex < 0;
2) beginIndex > endIndex;
3) endIndex больше чем длина строки

9.9. Кодировка по умолчанию

Код программы может быть записан в любой кодировке.

При компиляции программы осуществляется перекодировка из так называемой кодировки по умолчанию в Unicode.

Кодировка по умолчанию определяется операционной системой, в которой выполняется JVM.

Наиболее часто используемые кодировки в своих первых 128-и позициях содержат один и тот же набор символов (набор ASCII): английские буквы (большие и малые), десятичные цифры, символы пунктуации и т.д. Из этих символов составлено ядро языка java.

Однако в программе могут быть строковые литералы (которые в языковое ядро не входят), записанные при помощи символов с кодами большими чем 128.

В другой кодировке это будут уже другие символы и, соответственно, другие строковые литералы.

Замечание. Все символы внутри JVM кодируются при помощи Unicode. Проблема с кодировкой может возникнуть, когда символы читаются из внешнего по отношению к JVM источника: текстовый файл, сокет и т.д. Причем кодировка таких символов может отличаться от кодировки по умолчанию (подробнее см. раздел посвященный потокам ввода/вывода).

9.10. Лексикографическое сравнение строк

Класс String содержит метод compareTo, который позволяет лексикографически сравнивать строку, на которой вызывается этот метод, со строкой, которая передается методу в качестве параметра.

public int compareTo(String another)

Метод возвращает следующие значения:

0, если строка this совпадает со строкой another;

отрицательное целое число, если this лексикографически *предшествует* строке another;

положительное целое число, если this лексикографически *следует за* строкой another.

В случае несовпадения строк результатом является целое число (со знаком), которое определяется следующим образом в зависимости от двух случаев:

1) строки *отличаются в какой то позиции* символом, пусть k – первая слева такая позиция, тогда возвращаемое значение равно

this.charAt(k)-another.charAt(k);

2) строки *имеют разную длину* таким образом, что одна строка является *подстрокой* другой, тогда возвращаемое значение равно

this.length()-another.length().

```
int k;

String s1 = "ABC";
String s2 = "ABE";
String s3 = "ABCDEF";

k = S1.compareTo(S2); // k = 'C' - 'E' = -2
k = S1.compareTo(S3); // k = 3 - 6 = -3
```

Замечание. Метод, который возвращает значение (т.е. не является void методом) можно вызывать без присваивания этого значения некоторой переменной. Для некоторых методов такой вызов лишен какого либо смысла, для других же просто игнорирует возвращаемое значение, которое является второстепенным результатом работы метода.

```
String s = "abc";
int k = s.compareTo("abf"); // k = -3
s.compareTo("abf"); // метод возвращает значение -3, которое никак не используется
```

9.11. Метод concat

Метод **concat** класса String *возвращает* строку this объединенную со строкой str.

public String concat(String str)

Метод выбрасывает исключение java.lang.NullPointerException если str = null.

Замечание. Вызов метода concat не меняет строку str.

```
String s1 = "123";
String s2 = "567";
s1.concat(s2);
System.out.print(s1); // будет выведено 123
```

9.12. Метод replace

Метод **replace** *возвращает* строку this заменяя в ней все вхождения символа oldChar на newChar.

public String replace(char oldChar, char newChar)

Замечание. Вызов метода replace не меняет строку str.

```
String s = "123";
s1.replace('1', 'A');
System.out.print(s); // будет выведено 123
```