String: mutable, immutable, StringBuilder, StringBuffer, StringTokinizer

- 1. Что такое mutable/imutable объекты и зачем они
- 2. Все классы, связанные со строками, что они делают и все их методы
- 3. Внутреннее устройство String, метод substring
- 4. Поиск, получение, удаление подстроки в String
- 5. Что можно делать с помощью класса StringBuilder, StringBuffer
- 6. Регулярные выражения, примеры
- 7. StringTokinizer, String.replace, String.split
- 8. Задачи

1. Что такое mutable/imutable объекты и зачем они

- Объекты, которые после создания можно изменить, называются изменяемыми или **mutable**. Если объекты после создания изменить нельзя, называются неизменяемыми или **immutable**.
- Два свойства, которые характерны практически для всех immutable объектов:
- **1) Неизменяемые объекты можно реализовать значительно проще, чем изменяемые.**
- **2) Неизменяемые объекты можно свободно использовать одновременно из разных нитей.**

- - А если мне нужно что-то поменять в таком объекте? Что вообще можно сделать с неизменяемым объектом?
- - Обычно immutable классы содержат различные методы, которые «как-бы» меняют объект, но вместо изменения самого объекта эти методы просто создают новый объект и возвращают его.
- **String s = "moscow";**
  String s2 = s.toUpperCase();
- Класс String – это immutable класс . **s** содержит строку «moscow», а **s2** –«MOSCOW» т.е **содержит новую строку**, которая идентична первой

2. Все классы, связанные со строками, что они делают и все их методы

- String отвечает за неизменяемую строку(immutable)
- **StringBuilder** – за изменяемую(mutable)
- StringBuffer – это копия StringBilder, но все методы которого объявлены **synchronized**
- Разве тут строка не изменяется?

Пример кода	Что происходит на самом деле

String s = "cat";
s= s + "-" + s;
	String s = "cat";
	StrinsBuilder s2 = new StringBuilder(s);
	s2.append("-");
	s2.append(s);
	s = s2.toString();

Метод(ы)	Пример(ы)

int **length**();	String s = "Good news everyone!";
	int n = s.**length**();
	int n = "Good news everyone!".**length**();

Метод **length** возвращает длину строки – количество в ней символов.	
char charAt**(int index)**	String s = "Good news everyone!"; char n = s.**charAt**(5); char n = "Good news everyone!". **charAt**(5);
Метод **charAt** возвращает символ строки по его номеру. Нумерация символов начинается с 0.	

char[]toCharArray **()**	String s = "Good news everyone!";
	for(char c: s.**toCharArray**()) {
	System.out.println(c);
	}

Метод toCharArray возвращает массив всех символов строки.	

Метод(ы)	Пример(ы)

boolean equals **(Object o)**	String s = "cat";
	boolean test1 = s.**equals**("cat");//true
	boolean test2 = s.**equals**("Cat");//false
	boolean test3 = s.**equals**("c"+"a"+"t");

Метод equals проверяет – совпадают ли строки	

boolean equalsIgnoreCase **(String str)**	String s = "cat";
	boolean test1 = s.**equalsIgnoreCase**("cat");
	boolean test2 = s.**equalsIgnoreCase**("Cat");
	boolean test3 = s.**equalsIgnoreCase**("cAT");

метод **equalsIgnoreCase** – совпадают ли строки, игнорируя регистр букв	

Метод(ы)	Пример(ы)
String toUpperCase**()**	String s = "Good news everyone!"; s = s.toUpperCase(); **Результат:** s == "GOOD NEWS EVERYONE!";
Метод **toUpperCase** возвращает копию строки, все символы которой – большие.	
String toLowerCase**()**	String s = "Good news everyone!"; s = s.toLowerCase(); **Результат:** s == "good news everyone!";
Метод **toLowerCase** возвращает копию строки, все символы которой – маленькие.	
String trim**()**	String s = "Good news everyone!"; s = s.trim();
Метод **trim** возвращает копию строки, без «пустых» символов в начале и конце.	

3. Внутреннее устройство String, метод substring

Метод(ы)	Пример(ы)
String substring**(int beginIndex, int endIndex)**	String s = "Good news everyone!"; s = s.**substring(1,6);**
	Результат: s == "ood n";
String substring**(int beginIndex)**	String s = "Good news everyone!"; s = s.**substring(1);**

- Метод **substring** возвращает часть строки
- Первый вариант возвращает подстроку, заданную начальным и конечным номерами символов. **Последний символ при этом не входит!**
- Второй вариант – от переданного номера и до конца строки.

Получение подстроки	Что хранится внутри подстроки
String s = "mama";	Что хранится в s: char[] **value** = {'m','a','m','a'}; **offset** = 0; **count** = 4;
Strins s2 = s.substring(1);	Что хранится в s2: char[] **value** = {'m','a','m','a'}; **offset** = 1; **count** = 3;
Strins s3 = s.substring(1, 3);	Что хранится в s3: char[] **value** = {'m','a','m','a'}; **offset** = 1; **count** = 2;

4. **Поиск, получение, удаление подстроки в String**

Метод(ы)	Пример(ы)
int indexOf**(String str)**	String s = "Good **n**ews everyone!"; int index = s.**indexOf ("ne");**
indexOf ищет в строке указанную строку с начала строки. Если найдена – возвращает номер ее первого символа, если нет - возвращает -1	**Результат:** index == 5
int indexOf**(String str, int from)**	String s = "Good news everyo**ne**!"; int index = s.**indexOf ("ne",7);**
indexOf ищет в строке указанную строку начиная с какого-то номера. Если строка найдена – возвращает номер ее первого символа, если нет - возвращает -1	**Результат:** index == 16

Метод(ы)	Пример(ы)
int lastIndexOf**(String str)**	String s = "Good news everyo**ne**!"; int index = s.**lastIndexOf("ne");**
lastIndexOf ищет указанную строку в строке с самого конца строки. Если строка найдена –возвращает номер ее первого символа, если не найдена - возвращает -1.	**Результат:** index == 16
int lastIndexOf**(String str,int from)**	String s = "Good **ne**ws everyone!"; int index = s.**lastIndexOf("ne",15);**
lastIndexOf ищет указанную строку в строке с конца, начиная с какого-то номера . Если строка найдена – возвращает номер ее первого символа, если не найдена - возвращает -1.	**Результат:** ?

Метод(ы)	Пример(ы)
String **replace**(char oldChar, char newChar)	String s = "G**oo**d news every**o**ne!"; String s2 = s.**replace**>('o', 'a');
Метод **replace** заменяет все вхождения определенного символа на другой.	**Результат:** s2 == "G**aa**d news every**a**ne!";
String **replaceAll**(String regex, String replacement)	String s = "Good **ne**ws everyo**ne**!"; String s2 = s.**replaceAll ("ne","_");**
Метод **replaceAll** заменяет все вхождения одной подстроки на другую.	**Результат:** s2 == "Good _ws everyo_!";
String **replaceFirst**(String regex, String replacement)	String s = "Good **ne**ws everyone!"; String s2 = s.**replaceFirst ("ne","_");**
Метод **replaceFirst** заменяет первое вхождение переданной подстроки на заданную подстроку.	**Результат:** s2 == "Good _ws everyone!";

5. Что можно делать с помощью класса **StringBuilder, StringBuffer**

- **1) У меня есть обычная строка, я хочу сделать ее изменяемой.**

```
String s = "Bender";
StringBuilder s2 = new StringBuilder(s);
```

- **2) Я хочу добавить символ к текущей «изменяемой строке»?**

```
String s = "Bender";
StringBuilder s2 = new StringBuilder(s);
s2.append("!");
```

- **3) А как преобразовать StringBuilder обратно в строку?**

```
String s = "Bender";
StringBuilder s2 = new StringBuilder(s);
s2.append("!");
s = s2.toString();
```

- **4) А если мне нужно удалить символ?**

  String s = "Bender";
  StringBuilder s2 = new StringBuilder(s);
  s2.**deleteCharAt(2);** //останется "Beder"

- **5) Я хочу заменить часть строки на другую?**

  String s = "Ben**de**r";
  StringBuilder s2 = new StringBuilder(s);
  s2.**replace (3, 5, "_DE_");** //будет "Ben_DE_r"

- **6) Мне нужно развернуть строку задом наперед?**

  String s = "Bender";
  StringBuilder s2 = new StringBuilder(s);
  s2.**reverse();** //будет "redneB";

6. Регулярные выражения

Шаблон	Описание	Примеры
.	Один любой символ	1
\d	Любая цифра	7
\D	Любая **не**цифра	C
\s	Пробел, перенос строки, символ табуляции	' '
\S	Что угодно, кроме пробела, табуляции, переноса строки	f
[a-z]	Любая буква от a до z	z
[0-9]	Любая цифра от 0 до 9.	8
\w	Любая буква	c
\W	Любая **не**буква	_

Шаблон	Описание	Примеры
[a-d]?	Символы a-d встречаются 0..1 раз	a, b, c, d
[b-d,z]+	Символы b,c,d,z встречаются 1.. ∞ раз	b, bcdcdbdbdbdbzzzzbbzbzb, zbz
[1,7-9]*	Символы 1,7,8,9 встречаются 0.. ∞ раз	1, 7, 9, 9777, 111199
1{5}	Символ 1 встречается 5 раз	11111
[1,2,a,b]{2}	Символы 1,2,a,b встречаются 2 раза	11, 12, 1a, ab, 2b, bb, 22
[a,0]{2,3}	Символы a,0 встречаются 2..3 раз	aa, a0,00,0a, aaa,000, a00,0a0, a0a

Шаблон	Описание	Примеры
A?	Символ A встречается 0..1 раз	A
B+	Символ B встречается 1.. ∞ раз	BBBB
C*	Символ C встречается 0.. ∞ раз	CCC
D{n}	Символ D встречается n раз	DDDD, для шаблона D{4}
E{n,}	Символ E встречается n.. ∞ раз	EEEEEEE, для шаблона E{2,}
F{n,m}	Символ F встречается n..m раз	EEEE, для шаблона E{2,4}

Регулярные выражения, примеры

- «^» - означает, что подстрока обязана включать начало строки.
- «$» - означает, что подстрока обязана включать конец строки.
- В регулярных выражениях символы «[ ] \ / ^ $. | ? * + ( ) { }» имеют специальное значение. Их еще называют «управляющие символы». Поэтому просто так их использовать в строке нельзя.
- Их необходимо экранировать. Для этого, используется символ «\».

Шаблон	Строка и найденные подстроки, совпадающие с шаблоном
a{3}	**aaa** a **aaa** a **aaa**
a{3}$	aaa a aaa a **aaa**
^a{3}	**aaa** a aaa a aaa
^a{3}$	aaa a aaa a aaa
\?{3}	aaa **???** aaa a

7. String.replace, String.split

Метод(ы)	Пример(ы)
boolean **matches**(String regex)	String s = "Good news everyone!"; Boolean test = s.**matches("news\\.*");**
Позволяет проверить, совпадает ли строка с шаблоном, заданным регулярным выражением	**Результат:** false (строка не начинается со слова news)
String[ ] **split**(String regex)	String s = "Good **ne**ws everyo**ne**!"; String[ ] ss = s.**split**("ne"); System.out.println(Arrays.toString(ss));
Разбивает строку на части, принимает маску подстроки-разделителя	**Результат (будет массив из трех строк):** [Good , ws everyo, !] "Good ", "ws everyo", "!";

Метод(ы)	Пример(ы)
String **replaceAll**(String regex, String replacement)	String s = "Good n**ew**s **ever**yon**e!**"; String s2 = s.**replaceAll ("e\\.","EX");**
Заменяет все вхождения одной подстроки на другу	**Результат:** s2 == "Good nEXs EXEXyonEX";
String **replaceFirst**(String regex, String replacement)	String s = "Good n**ew**s everyone!"; String s2 = s.**replaceFirst("e\\.","EX");**
Заменяет первое вхождение переданной подстроки на заданную подстроку	**Результат:** s2 == "Good nEXs everyone!";

7.StringTokinizer(еще один способ **разбиения строки на части**)

- Этот класс не использует регулярные выражения, вместо этого в него просто передается строка, состоящая из символов-разделителей. Преимущества этого подхода в том, что он не разбивает сразу всю строку на кусочки, а идет от начала к концу.
- Класс состоит из конструктора и двух методов. В конструктор нужно передать строку, которую мы разбиваем на части, и строку – набор символов, используемых для разделения.
- Метод nextToken возвращает очередной токен – подстроку.
- Метод hasMoreTokens() возвращает true, если еще остались не отданные подстроки.

Метод(ы)	Пример(ы)

boolean hasMoreTokens()
 	
String nextToken()
	String s = "Good **ne**ws everyo**ne**!";
	
	StringTokenizer tokenizer =
	  new **StringTokenizer**(s,"ne");
	while (tokenizer.**hasMoreTokens**())
	{
	 String token = tokenizer.**nextToken**();
	 System.out.println(token);
	}

	Вывод на экран будет таким: Good ws v ryo !

Задача 1 (com.simbirsoft.lesson00.task00)

- **Найти подстроку**
  Метод getPartOfString() должен возвращать подстроку начиная с символа после 1-го пробела и до конца слова, которое следует после 4-го пробела.
  Пример: "Курсы Simbirsoft - лучший способ прокачать уровень знаний Java."
  Результат: "Simbirsoft - лучший способ "
  На некорректные данные бросить исключение TooShortStringException (сделать исключением).

Задача 2

- **Между табуляциями**
  Метод getPartOfString() должен возвращать подстроку между первой и второй табуляцией.
  На некорректные данные бросить исключение TooShortStringException.

Задача 3

StringTokenizer

Используя StringTokenizer разделить query на части по разделителю delimiter.

Пример:
getTokens("level22.lesson13.task01", ".") == {"level22", "lesson13", "task01"}

ASCII (American Standard Code for Information Interchange)

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	0	NUL	32	20	(sp)	64	40	@	96	60	`
1	1	SOH	33	21	!	65	41	A	97	61	a
2	2	STX	34	22	"	66	42	B	98	62	b
3	3	ETX	35	23	#	67	43	C	99	63	c
4	4	EOT	36	24	$	68	44	D	100	64	d
5	5	ENQ	37	25	%	69	45	E	101	65	e
6	6	ACK	38	26	&	70	46	F	102	66	f
7	7	BEL	39	27	'	71	47	G	103	67	g
8	8	BS	40	28	(	72	48	H	104	68	h
9	9	TAB	41	29	)	73	49	I	105	69	i
10	A	LF	42	2A	*	74	4A	J	106	6A	j
11	B	VT	43	2B	+	75	4B	K	107	6B	k
12	C	FF	44	2C	,	76	4C	L	108	6C	l
13	D	CR	45	2D	-	77	4D	M	109	6D	m
14	E	SO	46	2E	.	78	4E	N	110	6E	n
15	F	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	60	3C	<	92	5C	\	124	7C	|
29	1D	GS	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	62	3E	>	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F	_	127	7F	DEL

```
package com.javarush.test.level18.lesson10.bonus01
```

Шифровка

Задача: Придумать механизм шифровки/дешифровки

Программа запускается с одним из следующих наборов параметров:
-e fileName fileOutputName
-d fileName fileOutputName
где
fileName - имя файла, который необходимо зашифровать/расшифровать
fileOutputName - имя файла, куда необходимо записать результат шифрования/дешифрования
-e - ключ указывает, что необходимо зашифровать данные
-d - ключ указывает, что необходимо расшифровать данные