Review - Strings and Regular Expressions

Special Characters

\\	backslash character
\t	tab character
\n	newLine character
\r	carriage return character
\f	form feed character
\a	alert (bell) character
\c*x*	control character *x*

Predefined character classes

.	Any character (may or may not match a line terminator)
\d	A digit: [0-9]
\D	A non-digit: [^0-9]
\s	A whitespace character: [\t\n\x0B\f\r]
\S	A non-whitespace character: [^\s]
\w	A word character: [a-zA-Z_0-9]
\W	A non-word character: [^\w]

```
String s = "ab  cd";  // tab between b and c
String[] t = s.split("\t");  // splits on tab
for(int i=0;i<t.length;i++)
    System.out.println(t[i] + " ");
```

Output:
ab
cd

```
String s = "ab    cd";  // spaces or tab between b and c
String[] t = s.split("\\s");  // splits on any whitespace
for(int i=0;i<t.length;i++)
    System.out.println(t[i] + " ");
```

Output:
ab
cd

```
String s = "Now I  have   seen  it all";
String[] t = s.split("\\s");
for(int i=0; i<t.length; i++)
System.out.println("x" + t[i] + "x");
```

Output:
x**Now**x
x**I**x
xx
x**have**x
xx
xx
x**seen**x
xx
x**it**x
x**all**x

```
String s = "and I  believe!";
String[] t = s.split("\\S"); // a, n, d  give empty Strings
for(int i=0; i<t.length; i++)
System.out.println("x" + t[i] + "x");
```

Output:
xx
xx
xx
x x
x x

Leading matches generate empty strings
Contiguous matches generate empty strings
Trailing matches DO NOT generate empty strings

```
String s = "Now I  have   seen  it all. Really!";
String[] t = s.split("\\s+");       // + sign treats contiguous spaces as one space
for(int i=0; i<t.length; i++)
System.out.println("x" + t[i] + "x");
```

Output:
x**Now**x
x**I**x
x**have**x
x**seen**x
x**it**x
x**all.**x
x**Really!**x

```
String s = "a1b2c3";
String[] t = s.split("\\d");  // splits on digits 1, 2 and 3
for(int i=0;i<t.length;i++)
    System.out.println("x" + t[i] + "x");
```

Output:
x**a**x
x**b**x
x**c**x

```
String s = "a1b2c3";
String[] t = s.split("\\D");  // splits on chars a, b and c
for(int i=0;i<t.length;i++)
    System.out.println("x" + t[i] + "x");
```

Output:
xx
x**1**x
x**2**x
x**3**x

```
String s = "abc123";
String[] t = s.split("\\d");   // digits at end do not generate empty strings
for(int i=0;i<t.length;i++)
    System.out.println("x" + t[i] + "x");
```

```
Output:
xabcx
```

```
String s = "ab123c";
String[] t = s.split("\\d");   // digits 2 and 3 generate empty strings
for(int i=0;i<t.length;i++)
    System.out.println("x" + t[i] + "x");
```

```
Output:
xabx
xx
xx
xcx
```

```
String s = "ab123c";
String[] t = s.split("\\d+");   // digits 123  are treated as one
for(int i=0;i<t.length;i++)
    System.out.println("x" + t[i] + "x");
```

```
Output:
xabx
xcx
```

```
String s = "abc%123*";
String[] t = s.split("\\w");// splits on a, b, c,1,2,3  - empty strings for a, b, c, 2, and 3
for(int i=0;i<t.length;i++)
    System.out.println("x" + t[i] + "x");
```

```
Output:
xx
xx
xx
x%x
xx
xx
x*x
```

```
String s = "abc%123*";
String[] t = s.split("\\W");// splits on %, and *
for(int i=0;i<t.length;i++)
    System.out.println("x" + t[i] + "x");
```

```
Output:
xabcx
x123x
```

Character Classes

[abc]	a, b, or c (simple class)
[^abc]	Any character except a, b, or c (negation)
[a-zA-Z]	a through z or A through Z, inclusive (range)
+	count as one if contiguous
*	split once as empty string and then count as one if contiguous

```
String s = "abcdef";
String[] t = s.split("[be]");  // splits on b and e
for(int i=0;i<t.length;i++)
    System.out.println("x" + t[i] + "x");
```

```
Output:
xax
xcdx
xfx
```

```
String s = "abc def";
String[] t = s.split("[b e]");  // splits on b, space, and e
for(int i=0;i<t.length;i++)
    System.out.println("x" + t[i] + "x");
```

```
Output:
xax
xcx
xdx
xfx
```

```
String s = "abcdef";
String[] t = s.split("[ad]");  // splits on a and d
for(int i=0;i<t.length;i++)     // empty string before a
    System.out.println("x" + t[i] + "x");
```

split on a: empty string is before a and bcdef is after the a so **t[0] = ""**
Then split on d: bc before d and ef is after the d so **t[1] = bc**
Nothing left to split on so **t[2] = ef**

```
Output:
xx
xbcx
xefx
```

```
String s = "abc def";
String[] t = s.split("[a d]");  // splits on a, space, and d
for(int i=0;i<t.length;i++)     // empty string before a and d
    System.out.println("x" + t[i] + "x");
```

```
Output:
xx
xbcx
xx
xefx
```

```
String s = "abcdbcdef";
String[] t = s.split("[bd]"); // splits on each b and each d
for(int i=0;i<t.length;i++)
    System.out.println("x" + t[i] + "x");
```

```
Output:
xax
xcx
xx
xcx
xefx
```

```
String s = "abcdbcdef";
String[] t = s.split("[bd]+"); // treats db as one character
for(int i=0;i<t.length;i++)
    System.out.println("x" + t[i] + "x");
```

```
Output:
xax
xcx
xcx
xefx
```

```
String s = "abaabbadbda";
String[] t = s.split("[bd]+"); // treats contiguous b's and d's as one character
for(int i=0;i<t.length;i++)
          System.out.println("x" + t[i] + "x");
```

```
Output:
xax
xaax
xax
xax
```

```
String s = "abcdbcdef";
String[] t = s.split(" [^bd]"); // treats bcdcd as one character
for(int i=0;i<t.length;i++)
    System.out.println("x" + t[i] + "x");
```

```
Output:
xx
xbcdbcdx
```

```
String s = "babcdrcedf";
String[] t = s.split("[b-d]+"); // splits on b, c or d
for(int i=0;i<t.length;i++)
    System.out.println("x" + t[i] + "x");
```

```
Output:
xx
xax
xrx
xex
xfx
```

```
String s = "akbkckdkek";
String[] t = s.split("k",3); // splits into an array of length 3 on the first 2 k's
for(int i=0;i<t.length;i++)
    System.out.println("x" + t[i] + "x");
```

```
Output:
xax
xbx
xckdkekx
```

```
String s = "abcdbcdef";
String[] t = s.split("[cd]",3); // contiguous cd gives empty string
for(int i=0;i<t.length;i++)
    System.out.println("x" + t[i] + "x");
```

```
Output:
xabx
xx
xbcdefx
```

```
String s = "abcdbcdefmmncnabcr";
String[] t = s.split("[b-d]+",3); // splits on first 3 non-contiguous b, c or d
for(int i=0;i<t.length;i++)
    System.out.println("x" + t[i] + "x");
```

```
Output:
xax
xefmmnx
xnabcrx
```

```
String s, regex;
String[] t = s.split(regex);
for(int i=0; i<t.length; i++)
   System.out.println("x" + t[i] + "x");
```

s = "baa2aacab"	**s = "baa2aacab"**	**s = "baa2aacab"**
regex="[a]"	**regex="[a]+"**	**regex="[a]*"**
xbx xx x2x xx xcx xbx	xbx x2x xcx xbx	xx xbx xx x2x xx xcx xx xbx

s = "mnmogbnoo2omnnggmomn"	**s = "mnmogbnoo2omnnggmomn"**	**s = "mnmogbnoo2omnnggmomn"**
regex = "[m-o]"	**regex = "[m-o]+"**	**regex = "[m-o]*"**
xx xx xx xx xgbx xx xx x2x xx xx xx xggx	xx xgbx x2x xggx	xx xx xgx xbx xx x2x xx xgx xgx

s = "Java is fun"	**s = "Java is fun"**	**s = "Java is fun"**	**s = "Java is fun"**
regex = "[aeiou]"	**regex = "[aeiou]+"**	**regex = "[aeiou]*"**	**regex = "[aeiou]+"**
xJx xvx x x xs fx xnx	xJx xvx x x xs fx xnx	xx xJx xx xvx xx x x xx xsx x x xfx xx xnx	xJx xvx xsx xfx xnx

s = "COMPUTER"	**s = "COMPUTER"**	**s = "COMPUTER"**	**s = "COMPUTER"**
regex = "[AEIOU]+"	**regex = "[^AEIOU]+"**	**regex = "[AEIOU]*"**	**regex = "[AEIOU]+"**
xCx xMPx xTx xRx	xx xOx xUx xEx	xx xCx xx xMx xPx xx xTx xx xRx	xJx xvx xsx xfx xnx

Review #12 Strings and Regular Expressions

Consider the code:

```
String s, regex;
String[] t = s.split(regex);
```

or

```
String s, regex;
String[] t = s.split(regex, num);
```

Place the value of each element of the array `t` in the corresponding blank. Use `""` for the empty string, `_` for a space, and leave undefined cells blank.

	s	**regex** or **regex,num**	**elements in t**
1.	`"1 2 3"`	`" "`	____ ____ ____ ____ ____ ____ ____ ____ ____
2.	`"a b c"`	`"a"`	____ ____ ____ ____ ____ ____ ____ ____ ____
3.	`"a b c"`	`"\\s"`	____ ____ ____ ____ ____ ____ ____ ____ ____
4.	`"a b c"`	`" b"`	____ ____ ____ ____ ____ ____ ____ ____ ____
5.	`"broom"`	`"o+"`	____ ____ ____ ____ ____ ____ ____ ____ ____
6.	`"b12cd"`	`"\\d"`	____ ____ ____ ____ ____ ____ ____ ____ ____
7.	`"ac dc"`	`"\\S"`	____ ____ ____ ____ ____ ____ ____ ____ ____
8.	`"3x^5"`	`"[x^]"`	____ ____ ____ ____ ____ ____ ____ ____ ____
9.	`"diligent"`	`"[aeiou]"`	____ ____ ____ ____ ____ ____ ____ ____ ____
10.	`"av ce de jk"`	`"\\w"`	____ ____ ____ ____ ____ ____ ____ ____ ____
11.	`"at#no@time"`	`"\\W"`	____ ____ ____ ____ ____ ____ ____ ____ ____
12.	`"to be or not"`	`"o",2`	____ ____ ____ ____ ____ ____ ____ ____ ____
13.	`"a 12 abc 1234"`	`"\\d"`	____ ____ ____ ____ ____ ____ ____ ____ ____
14.	`"a 12 abc 1234"`	`"\\d+"`	____ ____ ____ ____ ____ ____ ____ ____ ____
15.	`"a 12 abc 1234"`	`"\\D+"`	____ ____ ____ ____ ____ ____ ____ ____ ____
16.	`"abcbdbeb"`	`"b",3`	____ ____ ____ ____ ____ ____ ____ ____ ____
17.	`"carpet"`	`"[^p-r]"`	____ ____ ____ ____ ____ ____ ____ ____ ____
18.	`"carpet"`	`"[p-r+]"`	____ ____ ____ ____ ____ ____ ____ ____ ____
19.	`"carpet"`	`"[p-r]+"`	____ ____ ____ ____ ____ ____ ____ ____ ____
20.	`"java"`	`"^a",2`	____ ____ ____ ____ ____ ____ ____ ____ ____
21.	`"weekday"`	`"[^ae]"`	____ ____ ____ ____ ____ ____ ____ ____ ____
22.	`"1a2b3c"`	`"\\D"`	____ ____ ____ ____ ____ ____ ____ ____ ____
23.	`"review #12"`	`"[ e]|\\d+"`	____ ____ ____ ____ ____ ____ ____ ____ ____
24.	`"a aa aaa aaaa"`	`"a+",3`	____ ____ ____ ____ ____ ____ ____ ____ ____
25.	`"banana split"`	`"[an ]+"`	____ ____ ____ ____ ____ ____ ____ ____ ____

Consider the code:

```
String s, regex;
String[] t = s.split(regex);
```

or

```
String s, regex;
String[] t = s.split(regex, num);
```

Place the value of each element of the array `t` in the corresponding blank. Use `""` for the empty string, `_` for a space, and leave undefined cells blank.

	s	**regex** or **regex,num**	**elements in t**
26.	`"graduation"`	`"[aeiou]+"`	____ ____ ____ ____ ____ ____ ____ ____ ____
27.	`"graduation"`	`"[aeiou]*"`	____ ____ ____ ____ ____ ____ ____ ____ ____
28.	`"pp qq rr ss t"`	`"[pr+]"`	____ ____ ____ ____ ____ ____ ____ ____ ____
29.	`"convex"`	`"/*"`	____ ____ ____ ____ ____ ____ ____ ____ ____
30.	`"convex"`	`"/.*"`	____ ____ ____ ____ ____ ____ ____ ____ ____
31.	`"abc@aol.com"`	`"."`	____ ____ ____ ____ ____ ____ ____ ____ ____
32.	`"abc@aol.com"`	`"[.]"`	____ ____ ____ ____ ____ ____ ____ ____ ____
33.	`"abc@aol.com"`	`"a+"`	____ ____ ____ ____ ____ ____ ____ ____ ____
34.	`"aaa@aaa.com"`	`"a*"`	____ ____ ____ ____ ____ ____ ____ ____ ____
35.	`"analyze"`	`"a-o", 3`	____ ____ ____ ____ ____ ____ ____ ____ ____