Lab 4:

- This lab assignment is composed by one tutorial and one exercise.
- *Using your personal login*, submit your source code to Mooshak, problems C and D:

  http://deei-mooshak.ualg.pt/~hdaniel

  up to March 30, 2020

Problem D: The extermination game

Submit your source code to Mooshak at:

http://deei-mooshak.ualg.pt/~hdaniel

up to Mar 30, 2020

Bugs have been invading our system.
There are small bugs, big bugs, sneaky bugs, illusive ones.
There are even jumping bugs … huh?

The **Debugger Master** have his hands full fighting them ... well actually he has no hands. To aid fighting the bugs, the Debugger Master summoned a legion of **worker processes**.

Using a distributed processing strategy the worker processes spreads across the system, waiting for instructions form the Debugger Master . The Debugger Master spots the bugs and sends their location to the worker processes. The worker processes move there and get us rid of the bugs.

However, jumping bugs, whatever this maybe, can change current location after realizing they were spotted. To keep operations silent, the Debugger Master encrypts the messages sent to the worker processes.

The bugs are fast and furio ... ehr ... well they are fast, but not very bright either. So a simple symmetric encryption will do.

Your task is ...
... to develop an **EncrypterSmith** process that will help the Debugger Master and the worker processes to communicate safely. For that you have to develop a program, able to encrypt and decrypt Strings. The encryption method uses string and character rotations, in a fashion somehow similar to the **WWII Enigma machine**, however is even simpler. It have just 2 passes, applied in sequence:

1) The string to be encrypted is rotated **n** characters, where **n** is an integer between [-9, 9]. Negative **n** rotates to the left and positve **n** to the right.

Examples:

```
if n = -3 string "ABCDEFGH"  will become  "DEFGHABC"  (rotate left) if n
=  2 string "ABCDEFGH"  will become  "GHABCDEF"  (rotate right)
```

2) To all characters' ascii codes are added a integer **p** between [-4,4], see ASCII table below:

Examples:

```
if p = -1 string "IBM:111"    will become "HAL9000" if p =
2 string "X15"          will become "Z37"
```

Pretty simple!... did I already mentioned that the bugs are not very bright?

Even so, to make it harder the decryption of messages by identifying repeating characters, **messages will NOT have spaces**.

But, … to help the worker processes decrypt the message, the Debugger Master will ensure that all encrypted **messages have the word "bug" at the beginning … but in any case**, ex: "bug", "Bug", "bUg", ...

So, to decrypt a message is enough to know the key and reverse both steps. The key have 2 integers: **n** and **p** as defined above.

Dec	Char	Dec	Char	Dec	Char	Dec	Char	Dec	Char
32	space	52	4	72	H	92	\	112	p
33	!	53	5	73	I	93	]	113	q
34	"	54	6	74	J	94	^	114	r
35	#	55	7	75	K	95	_	115	s
36	$	56	8	76	L	96	`	116	t
37	%	57	9	77	M	97	a	117	u
38	&	58	:	78	N	98	b	118	v
39	'	59	;	79	O	99	c	119	w
40	(	60	<	80	P	100	d	120	x
41	)	61	=	81	Q	101	e	121	y
42	*	62	>	82	R	102	f	122	z
43	+	63	?	83	S	103	g	123	{
44	,	64	@	84	T	104	h	124	|
45	-	65	A	85	U	105	i	125	}
46	.	66	B	86	V	106	j	126	~
47	/	67	C	87	W	107	k		
48	0	68	D	88	X	108	l		
49	1	69	E	89	Y	109	m		
50	2	70	F	90	Z	110	n		
51	3	71	G	91	[	111	o		

ASCII table

Constraints

Structure your code in 2 classes: **EncrypterSmith** and **Craftsman** each defined in its own file. **EncrypterSmith** class implements at least 3 methods:

```
//Encrypts string message with key passed in n and p
//returns the encrypted string static String
encrypt(int n, int p, String message);
//Tries to decrypt encriptedMessage with key passed in n and p
//returns the decrypted string static String decrypt(int n,
int p, String encriptedMessage);

//Tests all key combinations of n and p until the first letters
//in the string became "bug".
//returns found key in an array with 2 ints: n and p //word
is the word at the beginning of the string: "bug" static
int[] findKey(String encryptedMessage, String word);
```

Craftsman class reads input and uses **EncrypterSmith** craftmanship skills to encrypt or decrypt messages and generate the output.

The length of all messages is between 10 to 100 characters (does not need to be checked).

You should also develop unit tests in a separate file, for all the 3 methods defined above.
Each method should have at least 3 test cases.

Possible approach to decryption Use brute force. Today's computers are too fast for WWII Enigma machine like encryption.
findkey() can be implemented as:

```
for n in [-9, 9]
      for p in [-4, 4]
            decstr = decrypt(n, p, encryptedMessage)
            if lowercase(decstr[0-2]) == "bug" key has been found, break
```

Input
The input have 2 lines. The first line have 2 integers and the second line a string. If the numbers in the first line are both 0 0, the second line have a string to decrypt.
If the first line have values different than zero, they are the encryption key, the first is **n** and the second **p**. In these case the second line have a string to encrypt.

Output
Just one line with the decrypted or encrypted string, according to the input.

Input sample 0
2 2
bugIncacheMemory

Output sample 0
t{dwiKpecejgOgoq

Input sample 1
0 0
t{dwiKpecejgOgoq

Output sample 1
bugIncacheMemory

Input sample 2
0 0
Rd>Qppa@lkqoliibo_

Output sample 2

bUgATssdController