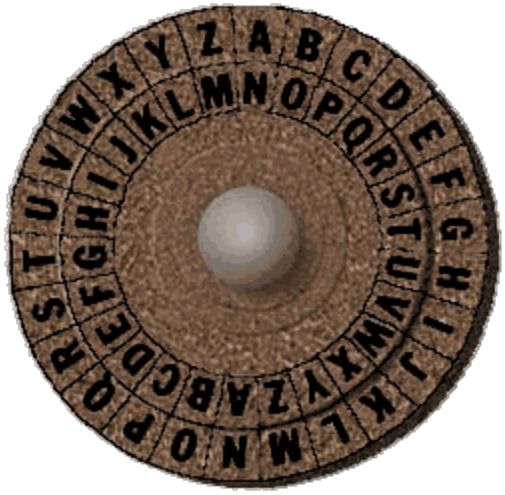
**CSCI 136 Programming Exam #1**
**Fundamentals of Computer Science II**
**Spring 2014**

This part of the exam is like a mini-programming assignment. You will create a program, compile it, and debug it as necessary. This part of the exam is open book and open web. You may use code from the course web site or from your past assignments. When you are done, submit all your Java source files to the Moodle exam #1 dropbox. Please ***double check you have submitted all the required files***.

You will have 100 minutes. No communication with any non-staff members is allowed. This includes all forms of real-world and electronic communication.

*Grading*. Your program will be graded on correctness and to a lesser degree on clarity (including comments) and efficiency. You will lose a substantial number of points if your code does not compile or crashes on typical inputs.



**Overview.** A Caesar cipher is a simple encryption technique in which each letter of the alphabet is replaced by another letter that is a fixed number of positions down the alphabet. For example, the plaintext letter "d" might be shifted three positions to the left to yield the ciphertext "a". Letters wrap around, so the plaintext letter "a" would be enciphered as "x", the plaintext letter "b" would be enciphered as "y", and so on. For example, encrypting the word "dab" with a Caesar cipher with a left shift of three, results in "axy".

To decrypt the message, you simply reverse and shift each letter three to the right. You can visualize a Caesar shift by aligning two alphabets, here is an alignment for a shift of three:

```
Plain:    abcdefghijklmnopqrstuvwxyz
Cipher:   xyzabcdefghijklmnopqrstuvw
```

We will assume the spaces in the original plaintext message have been left unencrypted, so using the above Caesar shift, here is an example plaintext and its corresponding ciphertext:

```
Plaintext:    this is a secret
Ciphertext:   qefp fp x pbzobq
```

You will be developing a program that decrypts a ciphertext such as the one above by trying all 26 possible shifts. This includes a shift of zero (i.e. the ciphertext is actually just the plaintext). Results will be scored by the percentage of English words each candidate decryption has. Start by downloading the file located at:

http://katie.mtech.edu/classes/csci136/caesar.zip

**Details.** The program `CaesarCipher` is a command-line program that takes two command-line arguments:

```
% java CaesarCipher
CaesarCipher <word file> <cipher file>
```

The first argument is the filename of a text file that contains a list of words that are known English words. Each line in the file contains a single word in lowercase and only has words with the letters a-z:

```
% more words.txt
a
aa
aaa
aaaas
aaadonta
aaahing
...
```

The second argument is the filename of a text file containing a single line of text. The text is a series of words that have been enciphered using a Caesar shift. Each word in the text is separated by a space. Only letters are enciphered, not spaces. Here is an example:

```
% more tale.txt
wh kog hvs psgh ct hwasg wh kog hvs kcfgh ct hwasg
```

**Part 1:** The `Words` class is responsible for loading a word list from a file. Clients can ask the `Words` object if a particular word is in the collection, how many words are in the collection, and for the percentage of words that are in-vocabulary in a given sentence. Here is the API (details appear in comments in the stub code):

```
class Words
-----------------------------------------------------------------
        Words(String filename)
    int getSize()
boolean inVocab(String word)
 double inVocabPercent(String sentence)
```

For full credit, your Words class should use a data structure providing constant time performance for the `inVocab()` method. The `CaesarShift` program should use the `Words` class to load the word list specified on the command-line and print out the number of words loaded. If the file cannot be read, `CaesarShift` should print an error message and terminate:

```
% java CaesarCipher words.txt tale.txt
Loaded 321666 words

% java CaesarCipher bogus.txt tale.txt
Word file not found!
```

**Part 2:** The `CaesarShift` program loads the ciphertext from the filename given as the second command-line argument. You can assume the file contains only a single line of ciphertext. All ciphered words consist of only the lowercase letters a-z. Words in the ciphertext are separated by spaces. Your `CaesarShift` program should print out the length of the ciphertext in characters (including spaces). If the file cannot be read, print an error message and terminate:

```
% java CaesarCipher words.txt tale.txt
Loaded 321666 words
Loaded cipher of length 50

% java CaesarCipher words.txt bogus.txt
Loaded 321666 words
Cipher file not found!
```

**Part 3:** You anticipate your program needing to decrypt some very long messages. Luckily you have a powerful server with four Xeon 8-core processors. In order to investigate all 26 possible decryptions simultaneously, you need to make your program multi-threaded. The `Shifter` class is responsible for investigating a particular shift on a ciphertext and calculating the percentage of words in the deciphered text that are in-vocabulary. Here is its API (further details are in comments in the stub code):

```
class Shifter implements Runnable
-----------------------------------------------------------------
       Shifter(Words words, String cipher, int shift)
void   run()
Result getResult()
```

In order to know its designated job, a `Shifter` object takes various parameters as input to its constructor. The `run` method is responsible for actually decrypting the ciphertext and computing the in-vocabulary percentage. The `getResult()` method is used to provide the final results to the main program. The `Result` object is a simple class that holds the resulting text, in-vocabulary percentage, and the shift that was used. Here is its API (further details are in comments in the stub code):

```
class Result
-----------------------------------------------------------------
       Result(String text, double inVocabPercent, int shift)
String toString()
```

Your `CaesarShift` program should fire up 26 threads, one for each possible shift. Your program should be constructed such that threads do their work in parallel (in practice for small ciphertexts, threads may finish before others threads get a chance to start). Once all threads have completed their work, print out the results from all shifts. At this point (part 3) results are not in any particular order (we'll be enforcing an ordering in part 4). In our output below they appear in ascending order of the shift. Note for output purposes, the in-vocabulary percent is shown to four decimal places and the deciphered text should be truncated to at most 40 characters of text:

```
% java CaesarCipher words.txt tale.txt
Loaded 321666 words
Loaded cipher of length 50
0.1667  0 wh kog hvs psgh ct hwasg wh kog hvs kcfg
0.3333  1 xi lph iwt qthi du ixbth xi lph iwt ldgh
0.1667  2 yj mqi jxu ruij ev jycui yj mqi jxu mehi
0.0000  3 zk nrj kyv svjk fw kzdvj zk nrj kyv nfij
0.1667  4 al osk lzw twkl gx laewk al osk lzw ogjk
0.3333  5 bm ptl max uxlm hy mbfxl bm ptl max phkl
0.0000  6 cn qum nby vymn iz ncgym cn qum nby qilm
0.3333  7 do rvn ocz wzno ja odhzn do rvn ocz rjmn
0.1667  8 ep swo pda xaop kb peiao ep swo pda skno
0.0000  9 fq txp qeb ybpq lc qfjbp fq txp qeb tlop
0.0000 10 gr uyq rfc zcqr md rgkcq gr uyq rfc umpq
0.1667 11 hs vzr sgd adrs ne shldr hs vzr sgd vnqr
1.0000 12 it was the best of times it was the wors
0.1667 13 ju xbt uif cftu pg ujnft ju xbt uif xpst
0.0000 14 kv ycu vjg dguv qh vkogu kv ycu vjg yqtu
0.0000 15 lw zdv wkh ehvw ri wlphv lw zdv wkh zruv
0.1667 16 mx aew xli fiwx sj xmqiw mx aew xli asvw
0.1667 17 ny bfx ymj gjxy tk ynrjx ny bfx ymj btwx
0.3333 18 oz cgy znk hkyz ul zosky oz cgy znk cuxy
0.3333 19 pa dhz aol ilza vm aptlz pa dhz aol dvyz
0.0000 20 qb eia bpm jmab wn bquma qb eia bpm ewza
0.0833 21 rc fjb cqn knbc xo crvnb rc fjb cqn fxab
0.3333 22 sd gkc dro locd yp dswoc sd gkc dro gybc
0.1667 23 te hld esp mpde zq etxpd te hld esp hzcd
0.1667 24 uf ime ftq nqef ar fuyqe uf ime ftq iade
0.1667 25 vg jnf gur orfg bs gvzrf vg jnf gur jbef
```

**Part 4:** You find it hard to spot the best decipherment in the result listing from part 3. Modify the `Result` class as well as the `CaesarShift` program so results are sorted in descending order of the in-vocabulary percentages. For results that have an identical in-vocabulary percentage, results in the set may appear in whatever order you like (e.g. the order within the `0.3333` results shown below is NOT important):

```
% java CaesarCipher words.txt tale.txt
Loaded 321666 words
Loaded cipher of length 50
1.0000 12 it was the best of times it was the wors
0.3333  1 xi lph iwt qthi du ixbth xi lph iwt ldgh
0.3333  5 bm ptl max uxlm hy mbfxl bm ptl max phkl
0.3333  7 do rvn ocz wzno ja odhzn do rvn ocz rjmn
0.3333 18 oz cgy znk hkyz ul zosky oz cgy znk cuxy
0.3333 19 pa dhz aol ilza vm aptlz pa dhz aol dvyz
0.3333 22 sd gkc dro locd yp dswoc sd gkc dro gybc
0.1667  0 wh kog hvs psgh ct hwasg wh kog hvs kcfg
0.1667  2 yj mqi jxu ruij ev jycui yj mqi jxu mehi
0.1667  4 al osk lzw twkl gx laewk al osk lzw ogjk
0.1667  8 ep swo pda xaop kb peiao ep swo pda skno
0.1667 11 hs vzr sgd adrs ne shldr hs vzr sgd vnqr
0.1667 13 ju xbt uif cftu pg ujnft ju xbt uif xpst
0.1667 16 mx aew xli fiwx sj xmqiw mx aew xli asvw
0.1667 17 ny bfx ymj gjxy tk ynrjx ny bfx ymj btwx
0.1667 23 te hld esp mpde zq etxpd te hld esp hzcd
0.1667 24 uf ime ftq nqef ar fuyqe uf ime ftq iade
0.1667 25 vg jnf gur orfg bs gvzrf vg jnf gur jbef
0.0833 21 rc fjb cqn knbc xo crvnb rc fjb cqn fxab
0.0000  3 zk nrj kyv svjk fw kzdvj zk nrj kyv nfij
0.0000  6 cn qum nby vymn iz ncgym cn qum nby qilm
0.0000  9 fq txp qeb ybpq lc qfjbp fq txp qeb tlop
0.0000 10 gr uyq rfc zcqr md rgkcq gr uyq rfc umpq
0.0000 14 kv ycu vjg dguv qh vkogu kv ycu vjg yqtu
0.0000 15 lw zdv wkh ehvw ri wlphv lw zdv wkh zruv
0.0000 20 qb eia bpm jmab wn bquma qb eia bpm ewza

% java CaesarCipher words.txt plain.txt
Loaded 321666 words
Loaded cipher of length 23
1.0000  0 this is not very secure
0.4000  6 znoy oy tuz bkxe ykiaxk
0.2000  2 vjku ku pqv xgta ugewtg
0.2000  5 ymnx nx sty ajwd xjhzwj
0.2000  7 aopz pz uva clyf zljbyl
0.2000  9 cqrb rb wxc enah bnldan
0.2000 16 jxyi yi dej luho iuskhu
0.2000 20 nbcm cm hin pyls mywoly
0.0000  1 uijt jt opu wfsz tfdvsf
```

```
0.0000  3 wklv lv qrw yhub vhfxuh
0.0000  4 xlmw mw rsx zivc wigyvi
0.0000  8 bpqa qa vwb dmzg amkczm
0.0000 10 drsc sc xyd fobi comebo
0.0000 11 estd td yze gpcj dpnfcp
0.0000 12 ftue ue zaf hqdk eqogdq
0.0000 13 guvf vf abg irel frpher
0.0000 14 hvwg wg bch jsfm gsqifs
0.0000 15 iwxh xh cdi ktgn htrjgt
0.0000 17 kyzj zj efk mvip jvtliv
0.0000 18 lzak ak fgl nwjq kwumjw
0.0000 19 mabl bl ghm oxkr lxvnkx
0.0000 21 ocdn dn ijo qzmt nzxpmz
0.0000 22 pdeo eo jkp ranu oayqna
0.0000 23 qefp fp klq sbov pbzrob
0.0000 24 rfgq gq lmr tcpw qcaspc
0.0000 25 sghr hr mns udqx rdbtqd
```

**% java CaesarCipher words.txt spanish.txt**

```
Loaded 321666 words
Loaded cipher of length 23
0.6667  1 poco a poco se va lejos
0.5000  5 tsgs e tsgs wi ze pinsw
0.5000  9 xwkw i xwkw am di tmrwa
0.5000 13 baoa m baoa eq hm xqvae
0.5000 21 jiwi u jiwi my pu fydim
0.3333  0 onbn z onbn rd uz kdinr
0.3333  3 rqeq c rqeq ug xc nglqu
0.3333  8 wvjv h wvjv zl ch slqvz
0.3333 11 zymy k zymy co fk votyc
0.3333 14 cbpb n cbpb fr in yrwbf
0.3333 15 dcqc o dcqc gs jo zsxcg
0.3333 16 edrd p edrd ht kp atydh
0.3333 18 gftf r gftf jv mr cvafj
0.3333 20 ihvh t ihvh lx ot exchl
0.3333 25 nmam y nmam qc ty jchmq
0.1667  2 qpdp b qpdp tf wb mfkpt
0.1667  4 srfr d srfr vh yd ohmrv
0.1667  6 utht f utht xj af qjotx
0.1667  7 vuiu g vuiu yk bg rkpuy
0.1667 10 yxlx j yxlx bn ej unsxb
0.1667 12 aznz l aznz dp gl wpuzd
0.1667 17 fese q fese iu lq buzei
0.1667 19 hgug s hgug kw ns dwbgk
0.1667 22 kjxj v kjxj nz qv gzejn
0.1667 23 lkyk w lkyk oa rw hafko
0.1667 24 mlzl x mlzl pb sx ibglp
```

```
% java CaesarCipher words.txt moby.txt
Loaded 321666 words
Loaded cipher of length 1142208
0.9795  7 loomings call me ishmael some years ago
0.2011 11 pssqmrkw gepp qi mwlqeip wsqi cievw eks
0.1995  0 ehhfbgzl vtee fx blaftxe lhfx rxtkl tzh
0.1989 13 ruusotmy igrr sk oynsgkr yusk ekgxy gmu
0.1932 14 svvtpunz jhss tl pzothls zvtl flhyz hnv
0.1775 17 vyywsxqc mkvv wo scrwkov cywo iokbc kqy
0.1677  1 fiigcham wuff gy cmbguyf migy syulm uai
0.1673 23 beecydwi sqbb cu yixcqub iecu ouqhi qwe
0.1658 22 addbxcvh rpaa bt xhwbpta hdbt ntpgh pvd
0.1578 19 xaayuzse omxx yq uetymqx eayq kqmde msa
0.1514  8 mppnjoht dbmm nf jtinbfm tpnf zfbst bhp
0.1512 20 ybbzvatf pnyy zr vfuznry fbzr lrnef ntb
0.1385  2 gjjhdibn xvgg hz dnchvzg njhz tzvmn vbj
0.1312 21 zccawbug qozz as wgvaosz gcas msofg ouc
0.1309  3 hkkiejco ywhh ia eodiwah okia uawno wck
0.1253  6 knnlhmfr bzkk ld hrglzdk rnld xdzqr zfn
0.1027 12 qttrnslx hfqq rj nxmrfjq xtrj djfwx flt
0.0912 15 twwuqvoa kitt um qapuimt awum gmiza iow
0.0892 18 wzzxtyrd nlww xp tdsxlpw dzxp jplcd lrz
0.0746 25 dggeafyk usdd ew akzeswd kgew qwsjk syg
0.0727 24 cffdzexj trcc dv zjydrvc jfdv pvrij rxf
0.0704 10 orrplqjv fdoo ph lvkpdho vrph bhduv djr
0.0633  9 nqqokpiu ecnn og kujocgn uqog agctu ciq
0.0568  4 illjfkdp zxii jb fpejxbi pljb vbxop xdl
0.0548 16 uxxvrwpb ljuu vn rbqvjnu bxvn hnjab jpx
0.0417  5 jmmkgleq ayjj kc gqfkycj qmkc wcypq yem
```

P.S. Despite the last ciphertext having over a million letters, it only takes about 4 seconds on my computer.