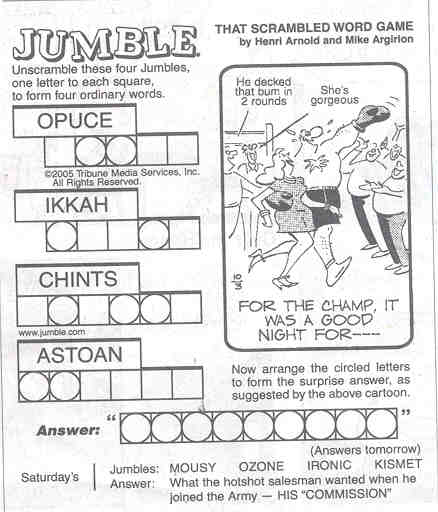
**Program #2. Solve Jumbled words**



[**Program2.java**](http://www.cs.pitt.edu/~hoffmant/401/program-02/Program2.java)**Starter File**

Your program should take the dictionary and the jumbles file on the command line like below so that I can test with smaller files.

C:\>java Program2 dictionary.txt jumbles.txt

* [**small-dictionary.txt**](http://www.cs.pitt.edu/~hoffmant/401/program-02/small-dictionary.txt)
* [**small-jumbles.txt**](http://www.cs.pitt.edu/~hoffmant/401/program-02/small-jumbles.txt)
* [**dictionary.txt**](http://www.cs.pitt.edu/~hoffmant/401/program-02/dictionary.txt)
* [**jumbles.txt**](http://www.cs.pitt.edu/~hoffmant/401/program-02/jumbles.txt)

**UPDATE: If your program produces the same lines as below BUT does not sort them vertically - OR - if the dictionary words on each line are not sorted after the jumbled word, then you still get full 100% credit. If your output matches mine EXACTLY (sorted vert & horiz) then you get 105%**

Here is correct output for the small files: (lines do not have to be in this exact order. Dictionary words on the line (after the jumbled word) do not have to be in alphabetical order either

arpt: part tarp trap

atc: act cat tac

atr: art rat tar

gdo: dog god

grof: frog

otsp: opts post pots spot stop tops

You must have your entire main wrapped in a try/catch block so that any possible Exception is caught. That exception must print "EXCEPTION DETECTED" + e then exit. If you hand in a file that does not trap an Exception - and you crash my script with that Exception, you will be deducted 10% that cannot be recoved in subsequent submissions!

Your program will read a sequence of jumbled words from a file. For each jumbled word, it will find all the real [**dictionary.txt**](http://www.cs.pitt.edu/~hoffmant/401/program-02/dictionary.txt) words that can be formed by unscrambling that jumbled word. When I test your program I will use a file named [**jumbles.txt**](http://www.cs.pitt.edu/~hoffmant/401/program-02/jumbles.txt). When given this jumbles file, your program should produce this output below:

addej: jaded

ahicryrhe: hierarchy

alvan: naval

annaab: banana

baltoc: cobalt

braney: nearby

celer: creel

couph: pouch

cukklen: knuckle

dica: acid cadi caid

dobeny: beyond

dobol: blood

dufil: fluid

dupled: puddle

eslyep: sleepy

ettniner: renitent

ettorp: potter

genjal: jangle

gluhc: gulch

hartox: thorax

hoybis: boyish

hucnah: haunch

iddec: diced

irrpo: prior

kutbec: bucket

lappor: poplar

lasia: alias

laurib: burial

lubly: bully

meefal: female

milit: limit

mopsie: impose

mycall: calmly

nekel: kneel

nokte: token

noper: prone

nwae: anew wane wean

nyegtr: gentry

perrim: primer

preko: poker

pudmy: dumpy

pypin: nippy

rebisc: scribe

rodug: gourd

rpeoims: imposer promise semipro

shewo: howes whose

wardty: tawdry

warllc:

yaldde: deadly

**Note that the jumbled word "warllc" has no matching word in the dictionary, thus you write nothing after it and go to the next word. You should also notice that the jumbled words are being listed in alphabetical order and so are the dictionary words after it.**

**Implementation Details**

* Read the strings from the dictionary file process the dictionary according to one of the strategies discussed in class or use your own algorithm.
* You are **not** allowed to use HashMap, ArrayList, TreeList, TreeSet or any other templated container class to store or process your Strings.
* You **are** allowed to use the Arrays class to sort and search your underlying array(s).

**Here is a sample program that produces a sorted version of a single String**

**import java.io.\*;**

**import java.util.\*;**

**public class Sort**

**{**

**public static void main( String args[] )**

**{**

**String unsorted = "zebra";**

**char cArr[] = unsorted.toCharArray();**

**Arrays.sort( cArr );**

**String sorted = new String( cArr );**

**System.out.println( "unsorted: " + unsorted + ", sorted: " + sorted );**

**}**

**}**