CSE143 Section #14 Problems

For this section, we will write methods that look for combinations of strings that constitute words. We will use a class called Dictionary with the following public methods:

Dictionary()	constructs a Dictionary from a dictionary file
contains(word)	returns whether the given word is in the dictionary
containsPrefix(prefix)	returns whether the dictionary has at least one word that begins with the given prefix
size()	returns the number of words in the dictionary
wordList()	returns a view of the words as a List<String>

We will use only the contains and containsPrefix methods in the code we write.

In addition to the dictionary, we will work with a list of strings that we will use to try to build up words. As an example, we will consider using the 2-letter state abbreviations used by the postal service. Below is a list of the state abbreviations:

AL, AK, AZ, AR, CA, CO, CT, DE, FL, GA, HI, ID, IL, IN, IA, KS, KY, LA,
ME, MD, MA, MI, MN, MS, MO, MT, NE, NV, NH, NJ, NM, NY, NC, ND, OH, OK,
OR, PA, RI, SC, SD, TN, TX, UT, VT, VA, WA, WV, WI, WY

These abbreviations can sometimes be combined to form words. For example, the state abbreviations for AL, MO, and ND can be combined to form the word ALMOND.

1. Write a recursive method called printJumbles that takes a Dictionary, a list of strings, and the number of strings to combine and that prints out all combinations with that number of strings that are in the dictionary:

```
public static void printJumbles(Dictionary words, List<String> options,
                                int choices) {
    ...
}
```

   For example, if passed the 50 state abbreviations as the options and 3 as the number of choices, it should print the following combinations:

ALARMS, ALCADE, ALKYNE, ALMOND, ALPACA, ARCADE, ARCANE, ARGALA, CALAMI,
CAMAIL, CANDID, CANDOR, COCAIN, CODEIN, CODEIA, COMADE, CONDOR, DECADE,
DECANE, DECOCT, DECODE, DEGAME, DEGAMI, DEMAND, DEMODE, DERIDE, FLORAL,
FLORID, FLORIN, GAMINE, INARMS, INCOME, INCONY, INDENE, INLAID, INLAND,
INVADE, INWIND, LARINE, LASCAR, LAWINE, MESCAL, MACACO, MARINE, MISCUT,
MISDID, MODEMS, NECTAR, ORDEAL, PALAPA, PAPAIN, PASCAL, PAVANE, RICTAL,
SCALAR, SCILLA, SCORIA, SCRIMS, VAHINE, VANDAL, WAHINE

2. Write a variation of the previous method called printJumbles2 that takes just the dictionary and the list of strings to combine and that finds all combinations of the strings of any length. This will require recognizing when a particular combination is a dead-end. For example, there are words in the dictionary that begin with the state abbreviation AL, but there are no words in the dictionary that begin with the state abbreviation NM.

   To solve this version, you should use the containsPrefix method of the Dictionary to see if a combination of strings has any chance of generating a word. As with the 8 queens solution, your recursive exploration method should have a precondition that this is not a dead end. In particular, you should guarantee that any string passed to the method is a prefix of at least one word in the dictionary.

   Using this approach, you should print the following words:

   AL, ALAR, ALARMS, ALCADE, ALGA, ALKY, ALKYNE, ALME, ALMA, ALMS, ALMOND, ALPACA, AKIN, AR, ARAK, ARCADE, ARCANE, ARCO, ARGALA, ARID, ARIL, ARIA, ARKS, ARMS, ARMORIAL, CACA, CACONYMS, CADE, CAID, CAIN, CAKY, CALAMARI, CALAMI, CALAMINE, CALAMONDIN, CAME, CAMAIL, CAMI, CAMS, CAMO, CANE, CANDID, CANDIDAL, CANDOR, CASCARILLA, CAVA, COAL, COCA, COCAIN, COCO, CODE, CODEIN, CODEIA, COIL, COIN, COKY, COLA, COME, COMA, COMADE, CONE, CONY, CONDOR, COOK, COWY, DE, DEAL, DEAR, DECADE, DECANE, DECO, DECOCT, DECODE, DEGAME, DEGAMI, DEIL, DEME, DEMAND, DEMO, DEMODE, DENE, DENY, DERIDE, DEVA, DEWY, FLAK, FLORAL, FLORID, FLORIN, GAGA, GAIN, GALA, GAME, GAMA, GAMINE, GAMS, GAMODEME, GANE, GANYMEDE, HI, HIDE, HILA, HIMS, HIND, ID, ILIA, ILKS, IN, INARMS, INCOME, INCONY, INDENE, INIA, INKS, INKY, INLAID, INLAND, INVADE, INWIND, KYAK, KYAR, LA, LADE, LAID, LAIN, LAKY, LAME, LAMA, LAMS, LANE, LAND, LARI, LARINE, LASCAR, LAVA, LAVALAVA, LAWINE, ME, MEAL, MEGA, MEGADEAL, MELAMINE, MEME, MEMS, MEMO, MEMORIAL, MEND, MESCAL, MESCALIN, MA, MAAR, MACACO, MADE, MAHIMAHI, MAID, MAIL, MAIN, MAINLAND, MALARIAL, MAMA, MAMS, MANE, MANY, MANDARIN, MANDORLA, MARINE, MAUT, MI, MICA, MIME, MINE, MIND, MIRI, MISCUT, MISDID, MO, MODE, MODEMS, MOIL, MOLA, MOME, MOMI, MOMS, MONY, MOOK, MOOR, MOORLAND, NE, NEAR, NECTAR, NEMA, NENE, OH, OHIA, OHMS, OK, OR, ORAL, ORCA, ORDEAL, PA, PACA, PACT, PAID, PAIL, PAIN, PAKS, PALAPA, PAMS, PANE, PAPA, PAPAIN, PASCAL, PAVANE, RIAL, RICTAL, RIDE, RIME, RIMS, RIND, SCALAR, SCAR, SCILLA, SCINCOID, SCORIA, SCRIMS, SCUT, UT, UTILIDOR, VAHINE, VAIL, VAIN, VANE, VANDAL, WADE, WAHINE, WAIL, WAIN, WAME, WANE, WANY, WAND, WIDE, WINE, WINY, WIND, WYND

3. Write a third version of the method called printJumbles3 that takes the same parameters as printJumbles2 but that doesn't allow any given string to be repeated in the result. For the state abbreviations, the following solutions should be rejected because they require using a state abbreviation more than once:

   CACA, COCO, DECADE, DECODE, DEMODE, DERIDE, GAGA, LAVALAVA, MEME, MAHIMAHI, MAMA, NENE, PALAPA, PAPA, PAPAIN

   To solve this version, include an extra parameter for your recursive exploration method that has a set of strings that have been used so far. With that extra information, you can guarantee that no string is used more than once for any given string you build up.