2. Consider the following class that creates a list of compound words:

```
public class CompoundWordCreator
{
   private List<String> wordList; //contains no duplicates

   /** @return true if word is in the dictionary; false otherwise
    */
   private boolean inDictionary(String word) {
      /* implementation not shown */
   }

   /** Combines all pairs of words in wordlist whose lengths sum to letterSum,
    *  and adds the new words to the list compoundWords if the new words were
    *  found in the dictionary. Words should not be combined with themselves.
    */
   private void addCompoundWords(List<String> compoundWords, int letterSum) {
      /* to be completed in part (b) */
   }

   /** precondition: wordList.size() > 0
    *  @return the length of the longest word in wordList
    */
   private int findMaxLength() {
      /* to be completed in part (a) */
   }

   /** precondition: wordList.size() > 0
    *  @return a list of compound words found in the dictionary that were created
    *          by combining strings in the list wordList
    *  postcondition: for each word, w, in list, inDictionary(w) == true and
    *                 3 <= w.length() <= findMaxLength().
    */
   public List<String> buildWords() {
      /* to be completed in part (c) */
   }

   // Constructors, other methods, and instance variables are not shown.
}
```

(a) Add a Method deleteString should delete all the specific string in wordlist and return the number of deleted string. For example {"~~cat~~ch a c~~at~~" , "w~~at~~ch TV "," time ","wh~~at~~ is in the h~~at~~?"} ,after running deleteString("at"), wordlist change to {"cch a c","wch TV","time","wh is in the h?"} and return 5.

```
/** precondition: wordList.size() > 0
 * @return the number of deleted string in wordList and update the wordList
 */
private int deleteString (String str) {
```

b) The list, `wordList`, contains a list of unique words that can be found in the dictionary. Method `addCompoundWords` creates two new words for each pair of words in `wordList` whose lengths add to `letterSum`. (For example, if "less" and "time" are chosen as a pair of words whose lengths sum to **8**, then the two new words created would be "lesstime" and "timeless.") Once the words are created, it checks to see if the new words are in the dictionary. Any new word that is found in the dictionary will be added to `compoundWords`. A word should not be combined with itself to create a new word.

Complete method `addCompoundWords`.

```
/** Combines all pairs of words in wordlist whose lengths sum to letterSum,
  * and adds the new words to the list compoundWords if the new words were
  * found in the dictionary. Words should not be combined with themselves
  */
private void addCompoundWords(List<String> compoundWords, int letterSum) {
```

(c) The `buildWords` method builds a list of new words. It finds these new words by considering all pairs of unique words from `wordList`. If their combined lengths are between 3 and the length of the largest word, inclusive, they are concatenated to create two new words. New words that can be found in the dictionary are added to `compoundWords`. You may use any methods found in the `CompoundWordCreator` class declaration. You may assume that anything you wrote in part (a) and part (b) works as intended.

Complete method `buildWords` below.

```
/** precondition: wordList.size() > 0
 *  @return a list of compound words found in the dictionary that were created
 *          by combining strings in the list wordList
 *  postcondition: for each word, w, in the returned list, inDictionary(w) == true and
 *                 3 <= w.length() <= findMaxLength().
 */
public List<String> buildWords() {
```