2. LAB B – BUILDER

Purpose

Given a text file with many words, we want to read it word by word and keep track of it in the following ways:

Create two builders:

1. Create a dictionary for these words, sorted alphabetically (eliminate duplicates)
2. Create a word counting map (for each word the number of times it appears in the text).

Lab Scenario (WHAT to do)	Implement the builder design pattern for a word processing application. Develop the following: 1. Base builder interface will be developed for easy maintenance 2. Concrete builders : - WordsCounterBuilder which counts words - DictionaryBuilder which sorts words 1 test class

Implementation Steps
(HOW to do it)
	Code the WordsBuilder interface
	`public void addWord(String word);`
	`public Collection getCollection();`
	
	Code the WordsCounterBuilder
	- extends WordsBuilder
	- contains a Map member variable
	(holds the words and their count)
	- implement addWord(String word)
	(method should update the count for the word added in the map)
	- implement getCollection
	(simply return the map keyset)
	
	Code the DictionaryBuilder
	- extend WordsBuilder
	- contains a TreeSet member variable
	- implement addWord
	(since TreeSet already knows how to sort words all we have to do is add the word)
	- implement getCollection
	(simply return the treeSet)
	
	Code the Test class
	Load a collection of strings into the builder and call getCollection().
	
	Iterate on the collection and print it.

Review of the Builder

4 classes will be created:

1. Interface WordsBuilder will serve as base of concrete Builders
2. WordsCounterBuilder is a concrete builder class which counts the appearances of each word added
3. DictionaryBuilder is a concrete builder which sorts the words added, and maintains only one of each word (no duplicates)
4. TextWordsDirector tests the 2 builders. It loads words (optionally from a file) and adds them to the builder.

2.1.1. Create a new interface – WordsBuilder

Serves as the base for concrete builders. Makes the application flexible; new concrete builders can be added later on

1. Create the interface WordsBuilder
   a. Implement public void addWord(String). Clients use this method to add words
   b. Implement public Collection getCollection(). Clients use this method to get the collection of words added different builders may return different collections

2.1.2. Create a new class WordsCounterBuilder

This builder will implement the WordsBuilder interface. The builder counts words added, by using a map data structure.

1. Create the class WordsCounterBuilder
2. Extend WordsBuilder
3. Implement a Map member variable

   Map will store each word with a counter
4. Implement the addWord(String) method

   Search for the word in the map
   - If the word isn't found, store it with a value of 1 indicating we have one of that word
   - If the word is found in the map, fetch its value, add 1 to it and store it (increasing the count for that word)
5. Implement the getCollection() method

   Return the map entrySet: EntrySet is a collection containing pairs : all the keys and values from the map

2.1.3. Create a new class DictionaryBuilder

This builder will implement the WordsBuilder. The builder will sort words added, and will prevent / duplicates

1. Create the class DictionaryBuilder
2. Extend WordsBuilder
3. Implement a TreeSet member variable: TreeSet is a sorted set data structure
4. Implement the addWord(String) method: Add the word to the TreeSet
5. Implement the getCollection() method: Return the TreeSet

2.1.4. Create the class TextWordsDirector

This class tests the builders we developed, by loading words, then adding them to the builder, and finally calling getCollection() on the builder and examining the results

1. Create the class TextWordsDirector
2. Create the main method
3. Test the WordsCountBuilder
    a. Create a WordsCountBuilder object
    b. Add some words to it
    c. Call getCollection and print the collection received
    d. Words should each have a count
4. test the DictionaryBuilder
    a. Create a DictionaryBuilder object
    b. Add some words to it
    c. Call getCollection and print the collection received
    d. Examine the output – words should be sorted and unique

Note: The solution for this exercise is available in the ‘solutions’ directory