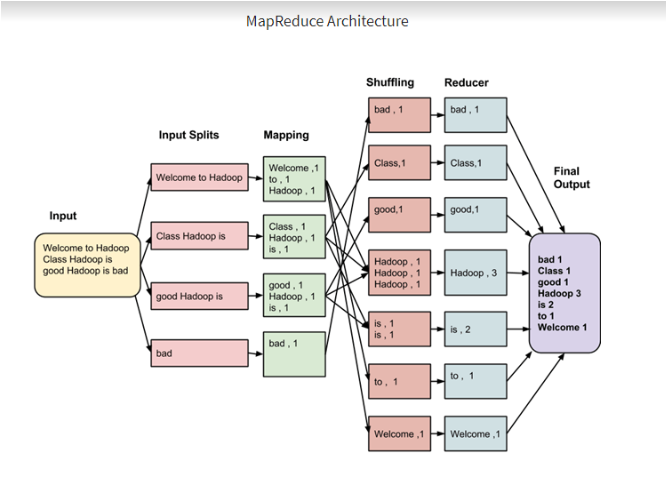
**ამოცანა- Map-Reduce**

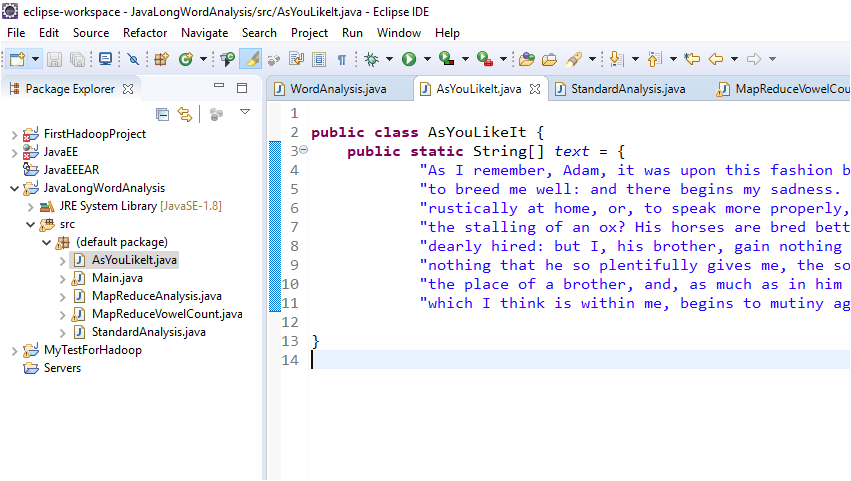


იმისათვის რომ კარგად გავერკვეთ თუ რა ამოცანის გადასაწყვეტად იყენებენ Hadoop-ის MapReduce ზმოცნას ამისათვის მოდით გავაკეთოთ ამოცანა ჯავას სტანდარტული ბიბლიოთეკების გამოყენებით. მოდით განვიხილოთ შემდეგი ტექსტი შექსპირის პიესიდან:

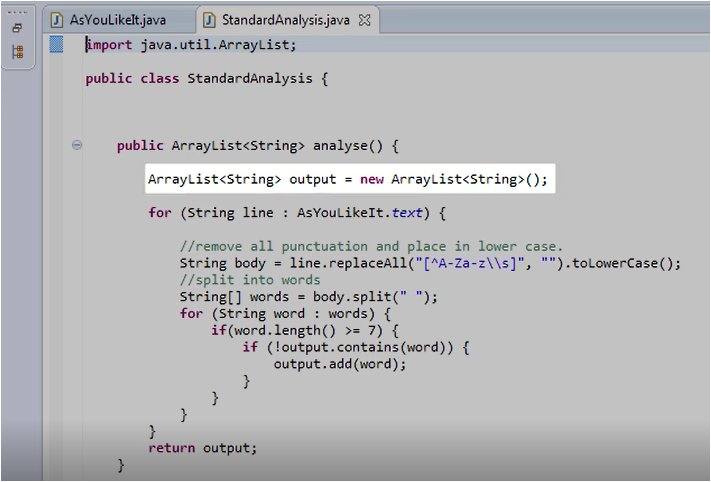
As I remember, Adam, it was upon this fashion bequeathed me by will but poor a thousand crowns, and, as thou sayest, charged my brother on his blessing to breed me well. And there begins my sadness. My brother Jaques he keeps at school, and report speaks goldenly of his profit. For my part, he keeps me rustically at home or, to speak more properly, stays me here at home unkept; for call you that “keeping” for a gentleman of my birth that differs not from the stalling of an ox? His horses are bred better, for, besides that they are fair with their feeding, they are taught their manage and, to that end, riders dearly hired. But I, his brother, gain nothing under him but growth, for the which his animals on his dunghills are as much bound to him as I. Besides this nothing that he so plentifully gives me, the something that nature gave me his countenance seems to take from me. He lets me feed with his hinds, bars me the place of a brother, and, as much as in him lies, mines my gentility with my education. This is it, Adam, that grieves me, and the spirit of my father, which I think is within me, begins to mutiny against this servitude. I will no longer endure it, though yet I know no wise remedy how to avoid it.

პირველ რიგში მოდით დავწეროთ პროგრამა რომელიც აღნიშნული ტექსტიდან ამოარჩევს ყველა იმ სიტყვას რომელშიც სიმბოლოების რაოდენობა მეტია ან ტოლი 7-ზე. ასეთ ერთ-ერთ სიტყვად ტექსტში გვევლინება სიტყვა - remember.

შევქმნათ სტანდარტული ჯავას პროექტი და მასში შევქმნათ კლასი AsYouLikeIt რომელიც შეიცავს სტრინგ ტიპის ელემენტების მქონე სტატიკურ მასივს სახელწოდებით *text* :



ჩვენ ასევე შევქმენით კლასი StandardAnalysis:

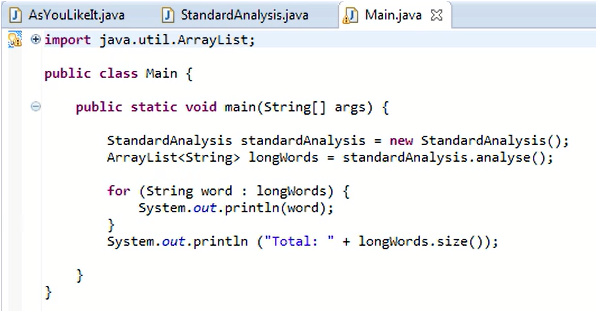


ჩვენი საძიებელი სიტყვების შესანახად ჩვენ ვქმნით ArrayList<String> ტიპის ახალ კოლექციას სახელწოდებით - output;

ხოლო საძიებელი სიტყვების მოსაძებნად კი ვიყენებთ შემდეგ ალგორითმს:

1. *text* მასივის ყოველი ელემენტისათვის რომელიც წარმოადგენს ჩვენი ტექსტის თითოეული სტრიქონს ვასრულებთ შემდეგ მოქმედებებს:
   1. აღნიშნული ელემენტიდან ამოვიღებთ ყველა სიმბოლოს გარდა ალფავიტის სიმბოლოსი და ერთმაგი “პრობელისა“;
   2. აღნიშნულ სტრიქონს, რომელიც ფაქტიურად ინახავს ერთ სტრიქონს რომელიც წარმოადგენს სიტყვებისაგან შექმნილ ტექსტს, გარდავქმნით ახალ მასივად რომლის ელემენტებიც იქნებიან აღნიშნული სიტყვები.
   3. ახალი მასივის ყოველი ელემენტისათვის შევასრულებთ შემდეგ მოქმედებებს:
      1. შევამოწმებთ ელემენტის სიგრძეს და თუ მისი სიგრძე მეტია ან ტოლი 7-ზე შევამოწმებთ კიდევ ერთ პირობას: შეიცავს თუ არა ამ ელემენტს output კოლექცია და თუ არ შეიცავს ამ შემთხვევაში დავუმატებთ ამ ელემენტს output კოლექციას
2. მეთოდი აბრუნებს output კოლექციას.

აღნიშნული კლასის გამოსაყენებლად ჩვენ პროექტის Main კლასს დავამატოთ შემდეგი

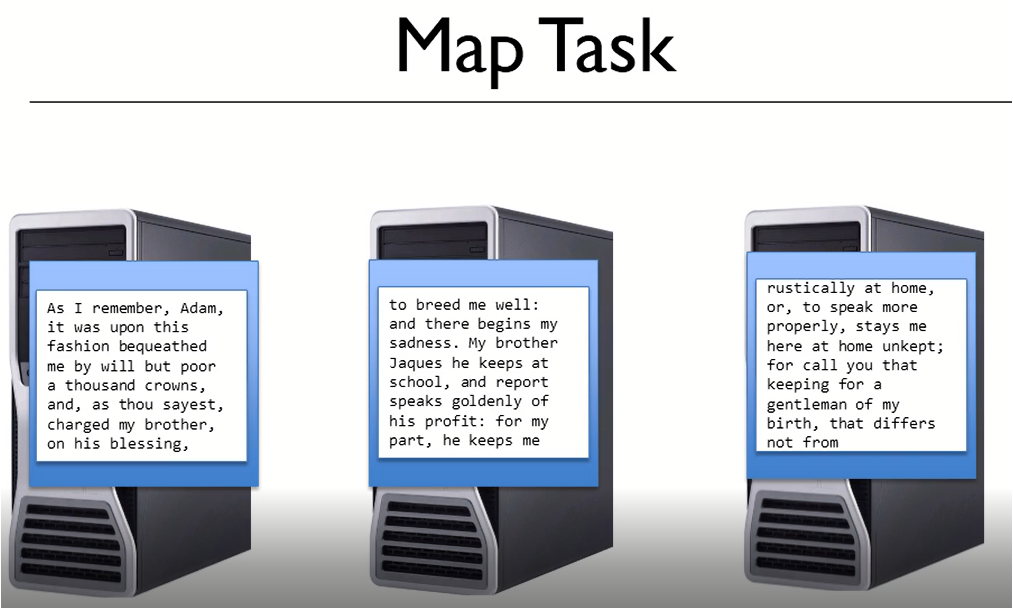


აღნიშნულ main მეთოდში ჩვენ ვახდენთ:

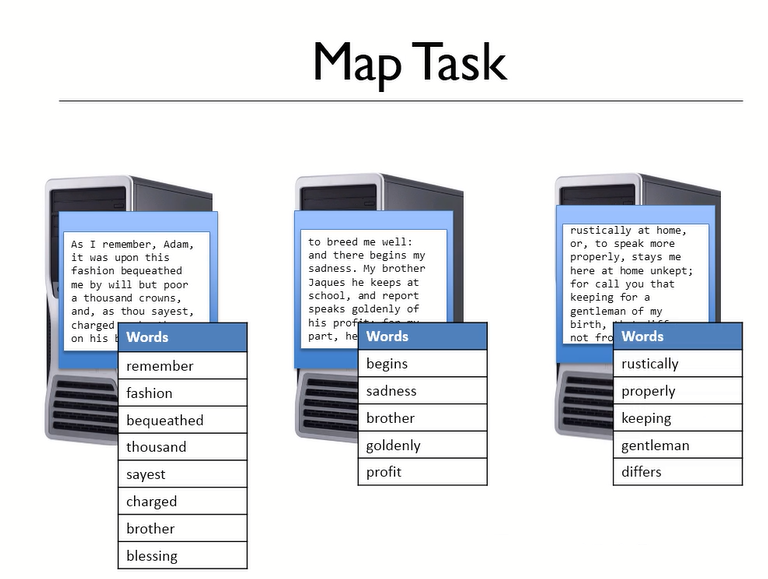
1. StandardAnalysis კლასის ახალი ეგზემპლიარის შექმნას;
2. StandardAnalysis კლასის analyse() მეთოდის გამოძახებას და კოლექციას რომელსაც აბრუნებს აღნიშნული მეთოდი ვინახავთ longWords ცვლადში;
3. ამის შემდეგ ვბეჭდავთ longWords კოლექციის თითოეულ ელემენტს და ბოლოს ელემენტების საერთო რაოდენობას კოლექციაში:

|  |  |
| --- | --- |
| remember | countenance |
| fashion | gentility |
| bequeathed | education |
| thousand | grieves |
| charged | against |
| brother | servitude |
| blessing | Total: 28 |
| sadness |  |
| goldenly |  |
| rustically |  |
| properly |  |
| keeping |  |
| gentleman |  |
| differs |  |
| stalling |  |
| besides |  |
| feeding |  |
| nothing |  |
| animals |  |
| dunghills |  |
| plentifully |  |
| something |  |

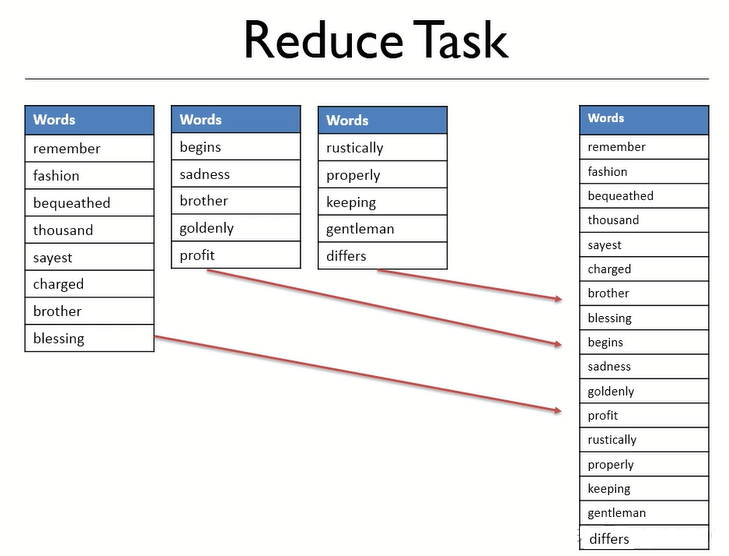
ამრიგად ჩვენ შევასრულეთ ტექსტში სიტყვების ანალიზის ამოცანა ჯავას სტანდარტული კლასების გამოყენებით. სანამ აღნიშნული ამოცანის რეალიზაციას Hadoop-ის სტანდარტული კლასების გამოყენებით დავიწყებდეთ განვიხილოთ თუ როგორ რეალიზდება ეს ამოცანა განაწილებული კლასტერის მიერ. როგორც ნახატიდან ჩანს ჩვენი ტექსტური მასივის თითოეული სტრიქონის დამუშავებას მოახდენს კლასტერში შემავალი ცალკეული სერვერი:



თითოეული სერვერის მიერ შესრულებული ამოცანის **(Map Task)** შედეგი იქნება აღნიშნული სერვერის მიერ დამუშავებული სტრიქონიდან აამორჩეული სიტყვები რომელთა სიგრძე მეტია ან ტოლი 7-ზე. აღნიშნული სერვერები მუშაობის დასრულების შედეგად დააბრუნებენ ArrayList<String> ტიპის მასივებს



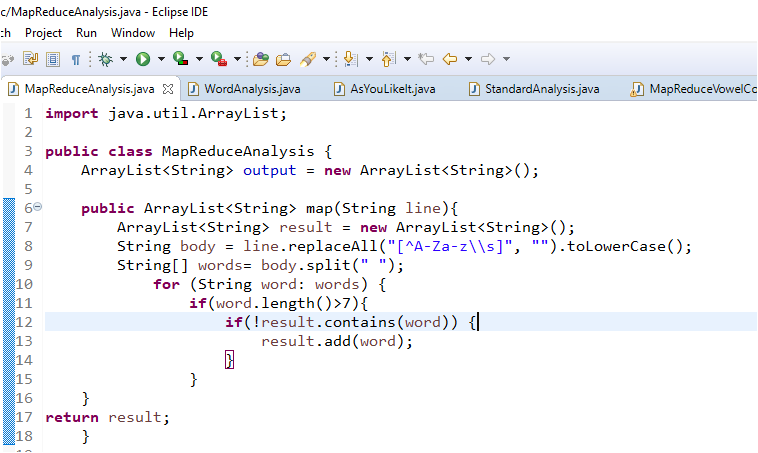
ჩვენი შემდეგი ამოცანაა (Reduce Task) თითოეული ამ მასივებიდან გავაერთიანოთ ერთ მასივში ისე რომ მიღებული მასივი არ შეიცავდეს დუბლიკატებს.



ამ მიზნით ჩვენ შევქმენით ახალი კლასი MapReduceAnalysis და ამ კლასში დავამატეთ მეთოდი

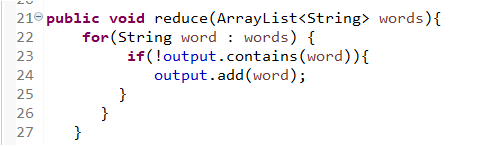
**public** ArrayList<String> map(String line)

აღნიშნულ მეთოდს გადაეცემა ერთი სტრიქონი სტრინგის სახით და იგი აბრუნებს ჩვენს მიერ საძიებელი სიტყვების მასივს



MapReduceAnalysis კლასში ჩვენ ასევე დამატებული გვაქვს მეთოდი

**public** **void** reduce(ArrayList<String> words)



რომელსაც ჩვენ შეგვიძლია გადავცეთ

public ArrayList<String> map(String line)

მეთოდის შესრულების შედეგად დაბრუნებული მასივი.

**public** **void** reduce(ArrayList<String> words)

მეთოდი სათითაოდ ამოიღებს მასივიდან ელემენტებს და იმ შემთხვევაში თუ ეს ელემენტი არ ეკუთვნის output კოლექციას დაამატებს მას აღნიშნულ კოლექციას.

ამრიგად ჩვენ ჯერ-ჯერობით განსაზღვრული გვაქვს ორი მეთოდი:

public ArrayList<String> map(String line)

**public** **void** reduce(ArrayList<String> words)

MapReduceAnalysis კლასში არსებულ ორ მეთოდს დავამატოთ კიდევ ერთი მეთოდი:

ArrayList<String> analyze()

რომელიც მოახდენს ცალკეული სერვერების მიერ შესასრულებელი ამოცანების იმიტაციას:

ArrayList<String> output1 =map(AsyYouLikeIt.text[0]);

ArrayList<String> output2 =map(AsyYouLikeIt.text[1]);

ArrayList<String> output3 =map(AsyYouLikeIt.text[2]);

ArrayList<String> output4 =map(AsyYouLikeIt.text[3]);

ArrayList<String> output5 =map(AsyYouLikeIt.text[4]);

ArrayList<String> output6 =map(AsyYouLikeIt.text[5]);

ArrayList<String> output7 =map(AsyYouLikeIt.text[6]);

ArrayList<String> output8 =map(AsyYouLikeIt.text[7]);

ArrayList<String> output9 =map(AsyYouLikeIt.text[8]);

ArrayList<String> output10 =map(AsyYouLikeIt.text[9]);

ArrayList<String> output11 =map(AsyYouLikeIt.text[10]);

ArrayList<String> output12 =map(AsyYouLikeIt.text[11]);

reduce(output1);

reduce(output2);

reduce(output3);

reduce(output4);

reduce(output5);

reduce(output6);

reduce(output7);

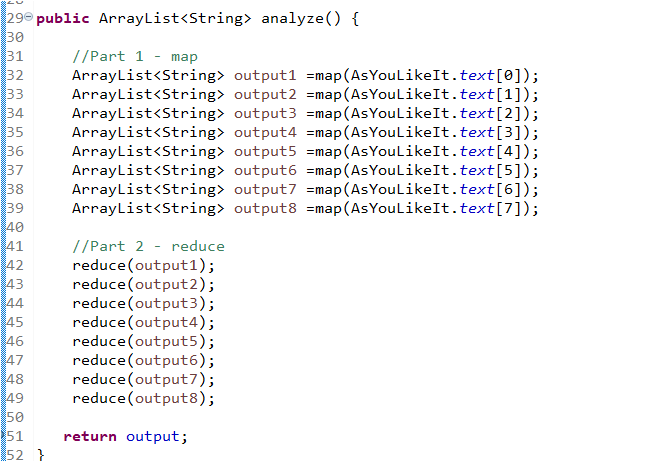
reduce(output8);

reduce(output9);

reduce(output10);

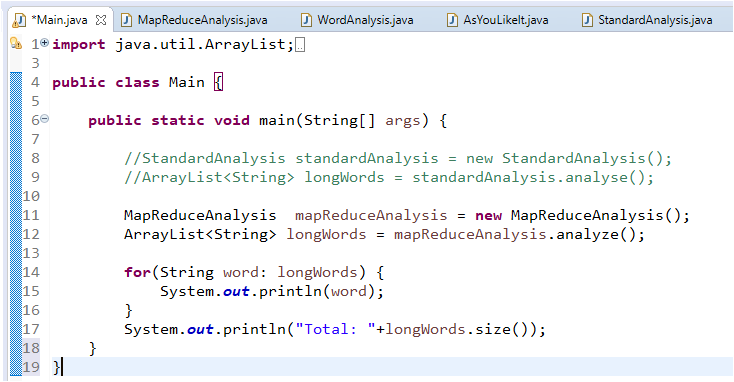
reduce(output11);

reduce(output1);

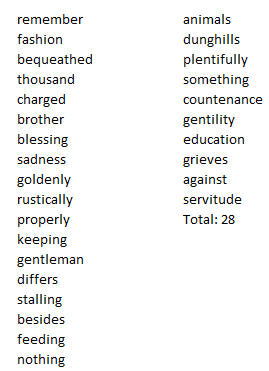


თვალსაჩინოებისათვის ჩვენ ცალკე გამოვყავით თითოეულისერვერის მიერ დასამუშავებელი ამოცანა ისე რომ, თითქოს ისინი ცალკე სერვერებზე ერთმანეთის პარალელურად სრულდებოდეს.

ამის შემდგომ შევცვალოთ Main კლასის main() მეთოდი. დავეკომენტაროთ ჩვენს მიერ ადრე შექმნილი StandardAnalysis კლასის გამოყენებით რეალიზებული კოდი და ამ კლასის ნაცვლად გამოვიყენოთ MapReduceAnalysis კლასი:



თუ შევასრულებთ აღნიშნულ კოდს დავინახავთ რომ მივიღებთ იგივე შედეგს:



ამასთან გასათვალისწინებელია ერთი გარემოება, კერძოდ ის რომ არ აქვს მნიშვნელობა თუ რა თანმიმდევრობით შევასრულებთ შემდეგ ამოცანებს:

//Part 1 - map

ArrayList<String> output1 =map(AsYouLikeIt.*text*[0]);

ArrayList<String> output2 =map(AsYouLikeIt.*text*[1]);

ArrayList<String> output3 =map(AsYouLikeIt.*text*[2]);

ArrayList<String> output4 =map(AsYouLikeIt.*text*[3]);

ArrayList<String> output5 =map(AsYouLikeIt.*text*[4]);

ArrayList<String> output6 =map(AsYouLikeIt.*text*[5]);

ArrayList<String> output7 =map(AsYouLikeIt.*text*[6]);

ArrayList<String> output8 =map(AsYouLikeIt.*text*[7]);

ჩვენ შეგვიძლია შევცვალოთ მათი შესრულების თანმიმდევრობა:

//Part 1 - map

ArrayList<String> output1 =map(AsYouLikeIt.*text*[0]);

ArrayList<String> output7 =map(AsYouLikeIt.*text*[6]);

ArrayList<String> output2 =map(AsYouLikeIt.*text*[1]);

ArrayList<String> output8 =map(AsYouLikeIt.*text*[7]);

ArrayList<String> output3 =map(AsYouLikeIt.*text*[2]);

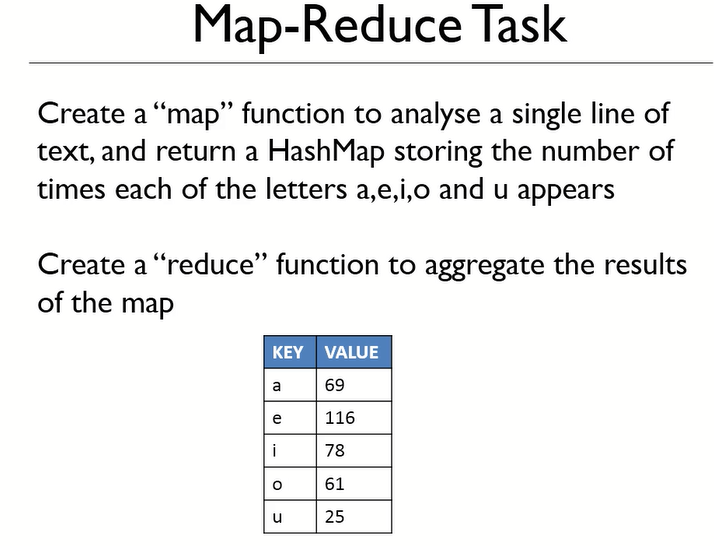
ArrayList<String> output6 =map(AsYouLikeIt.*text*[5]);

ArrayList<String> output4 =map(AsYouLikeIt.*text*[3]);

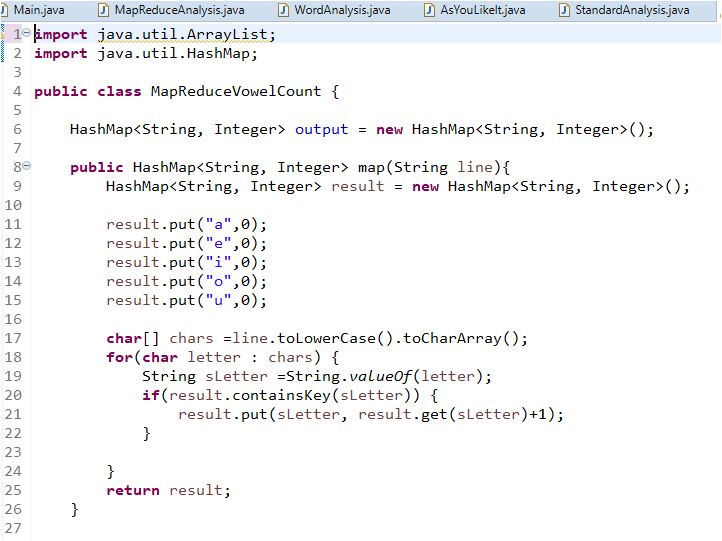
ArrayList<String> output5 =map(AsYouLikeIt.*text*[4]);

რის შედეგადაც მივიღებთ იგივე შედეგს.

იმისათვის რომჩვენ მივუახლოვდეთ Apache Hadoop - ის MapReduce კლასს ამისათვის საჭიროა რომ ჩვენი მეთოდები აბრუნებდნენ **HashMap** ტიპის ობიექტებს ნაცვლად  **ArrayList** ტიპის ობიექტებსა. ამისათვის განვიხილოთ ახალი ამოცანა:



შევადგინოთ პროგრამა რომელიც დაითვლის თუ რამდენჯერ გვხვდება ზემოთ მოყვანილი ხმოვნები *AsYouLikeIt* კლასის *text* მასივში. ამისათვის შევქმნათ ახალი კლასი *MapReduceVowelCount:*



აღნიშნულ *MapReduceVowelCount* კლასში ჩვენ შემოტანილი გვაქვს output კოლექცია

HashMap<String, Integer> output = **new** HashMap<String, Integer>();

აღნიშნული კოლექციიის ელემენტები წარმოადგენენ შემდეგი სახის წყვილებს:

"**key**/**value**"

გასაღების - "**key“** როლში ჩვენ განვიხილავთ საძიებელ ხმოვნებს ხოლო მნიშვნელობა "**value**" კი ინახავს იმ რაოდენობას რამდენჯერაც გვხვდება აღნიშული ხმოვანი ტექსტში.

**მეთოდი map(String line)**

აღნიშნულ კლასში ჩვენ ასევე შემოტანილი გვაქვს შემდეგი მეთოდი:

**public** HashMap<String, Integer> map(String line)

აღნიშნულ მეთოდს პარამეტრად გადაეცემა ჩვენი ტექსტის ერთი სტრიქონი. აღნიშნულ კლასში ჩვენ ასევე განსაზღვრული გვაქვს კოლექცია:

HashMap<String, Integer> result = **new** HashMap<String, Integer>();

რომელსაც დავაბრუნებთ მეთოდის შესრულების შედეგად.

**result** კოლექციაში ჩვენ თავიდან დავამატებთ შემდეგ ელემენტებს:

result.put("a",0);

result.put("e",0);

result.put("i",0);

result.put("o",0);

result.put("u",0);

ამის შემდგომ map(String line) მეთოდისათვის გადაცემულ პარამეტრის მნიშვნელობას (ტექსტის ერთ ხაზს) გარდავქმნით **char** ტიპის ელემენტების მასივად:

**char**[] chars =line.toLowerCase().toCharArray();

შემდგომ ეტაპზე ვაკეთებთ ციკლს, რომელიც მიღებული chars მასივის ყოველი ელემენტისატვის შეამოწმებს ეკუთვნის თუ არა ის chars კოლექციას (წარმოადგენს თუ არა ეს სიმბოლო ხმოვანს) და თუ აღმოჩნდა რომ იგი უკვე შესულია აღნიშნულ კოლექციაში როგორც გასარები მასინ ვამატებთ ახალ "**key**/**value**" წყვილს. ამასთან მნიშვნელობას ვზრდით ერთით:

**for**(**char** letter : chars) {

String sLetter =String.*valueOf*(letter);

**if**(result.containsKey(sLetter)) {

result.put(sLetter, result.get(sLetter)+1);

}

}

საბოლოოდ კი map მეთოდი აბრუნებს **result** კოლექციას.

**მეთოდი reduce(HashMap<String, Integer> letters)**

*MapReduceVowelCount* კლასში ჩვენ ასევე შემოტანილი გვაქვს **reduce()** მეთოდი. აღნიშნულ მეთოდს პარამეტრად გადაეცემა HashMap<String, Integer> ტიპის კოლექცია:

**public** **void** reduce(HashMap<String, Integer> letters) {

**for**(String key: letters.keySet()){

output.put(key, output.get(key)+letters.get(key));

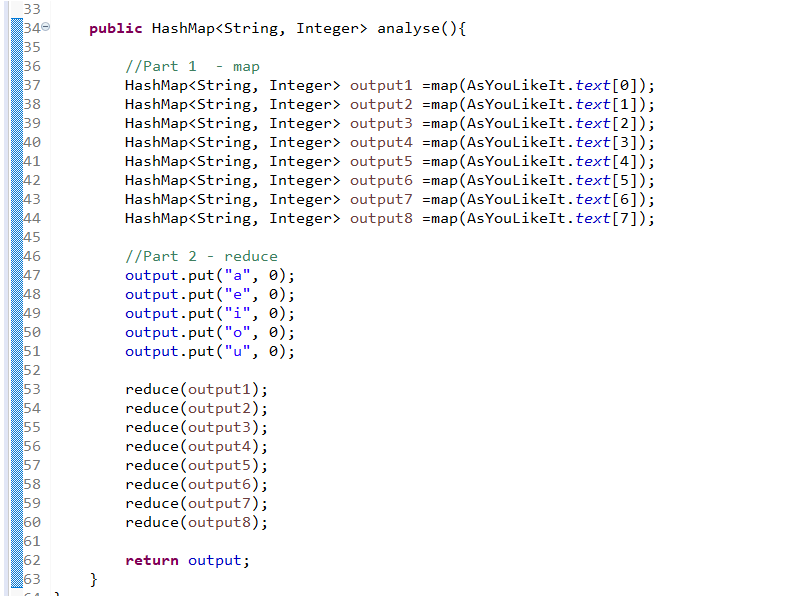
}

}

აღნიშნული მეთოდი ციკლის მეშვეობით გაივლის letters კოლექციის ყველა ელემენტს და იგი გაზრდის output კოლექციის key ინდექსის შესაბამის მნიშვნელობას letters.get(key) მნიშვნელობით.

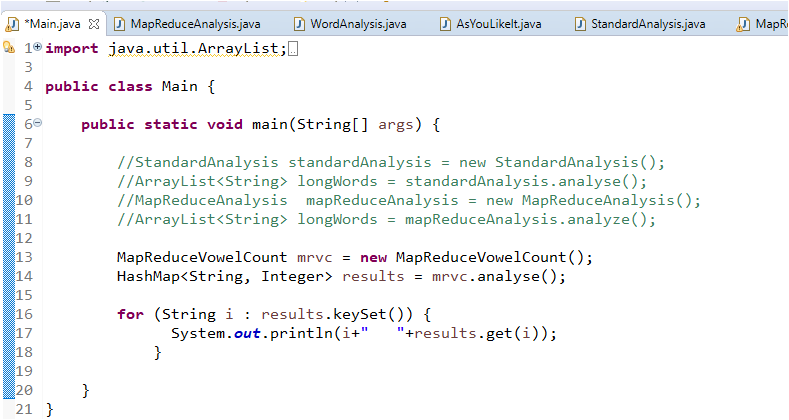
**მეთოდი analyse()**

*MapReduceVowelCount* კლასში ჩვენ ასევე შემოტანილი გვაქვს **analyse()** მეთოდი

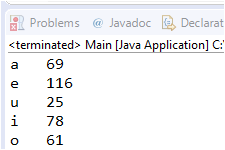


რომელიც ცალკ-ცალკე აანალიზებს ტექსტის თითოეულ ხაზს და აბრუნებს ტექსტში არსებულ ხმოვნებს და მათ რაოდენობას კოლექციის სახით.

საბოლოოდ კი გადავაკეთოთ Main კლასის main() მეთოდი

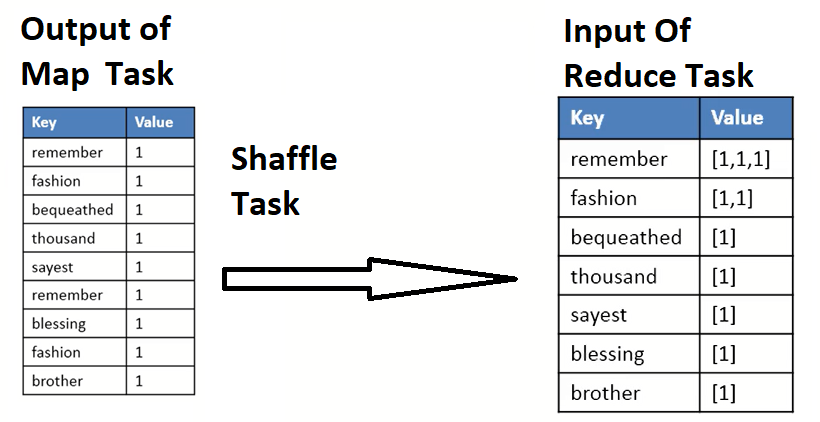


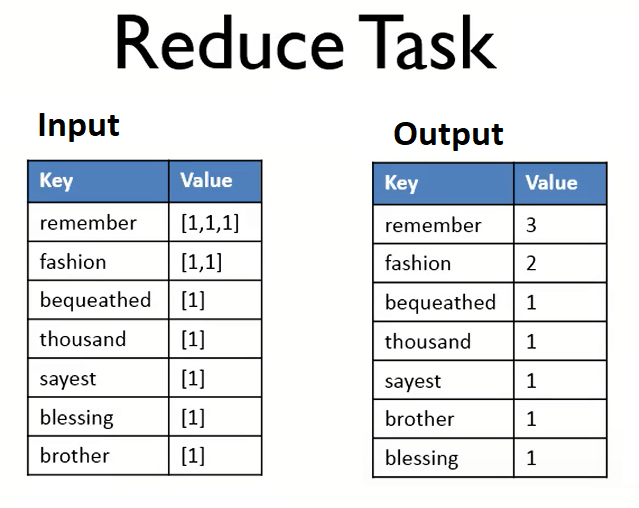
და შედეგად მივიღებთ შემდეგს:



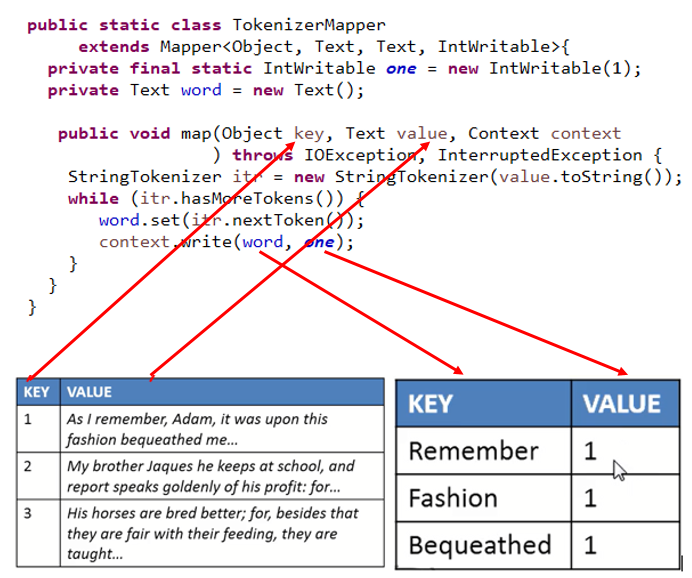
<https://www.srccodes.com/run-hadoop-wordcount-mapreduce-example-windows/>

**Hadoop Map Reduce Framework**

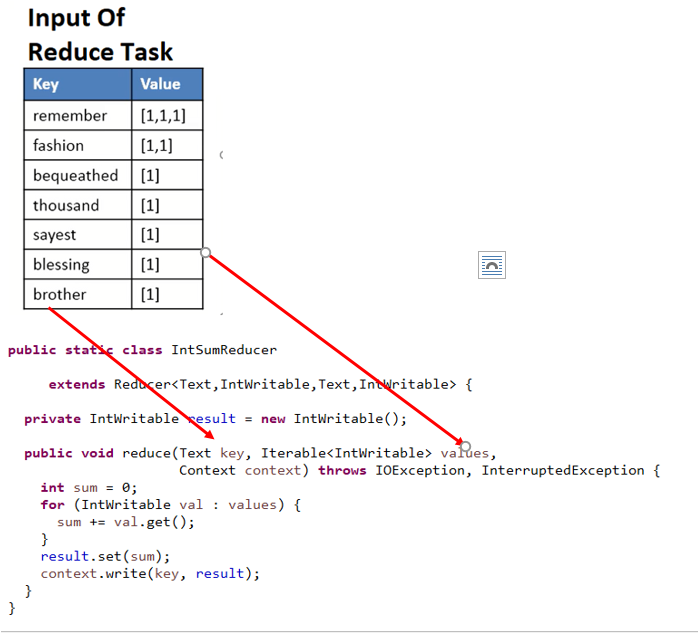




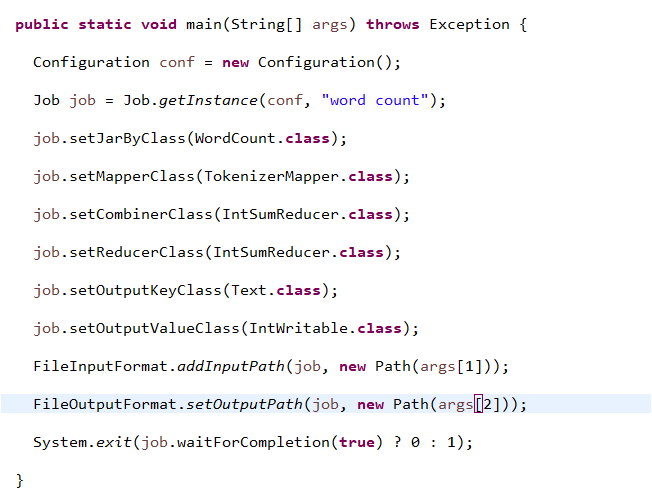
**Map Task**



**Reduce Task**



**Set Up Map Reduce Job Configuration**



Create jar file and Run the program (Map Reduce Job) :

**hadoop jar wcnew.jar WordCount /user/beka/input/wordcounttest.txt /user/beka/wordcount/output**

