**Reza Shisheie**

**Homework #3**

**Due: March 30th 2017**

Question 1:

**package** homework3\_1;

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

**public** **class** homework3\_1 {

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

List<Integer> arrayList = **new** ArrayList<>();

**int** n = 100000;

**for** (**int** i = 0; i<n; i++){

arrayList.add(1); // 1 is autoboxed to new Integer(1)

}

// Linked list traverse using iterator

**long** time\_1 = System.*currentTimeMillis*();

LinkedList<Object> linkedList = **new** LinkedList<>(arrayList);

ListIterator<Object> listIterator = linkedList.listIterator();

**for** (**int** i = 0; i<n; i++){

listIterator.next();

}

**long** time\_2 = System.*currentTimeMillis*();

System.***out***.print("The time for traversing using iterator is: ");

System.***out***.print(time\_2 - time\_1);

System.***out***.println(" milli seconds");

// Linked list traverse using get

time\_1 = System.*currentTimeMillis*();

**for** (**int** i = 0; i<n; i++){

linkedList.get(i);

}

time\_2 = System.*currentTimeMillis*();

System.***out***.print("The time for traversing using get is: ");

System.***out***.print(time\_2 - time\_1);

System.***out***.println(" milli seconds");

}

}

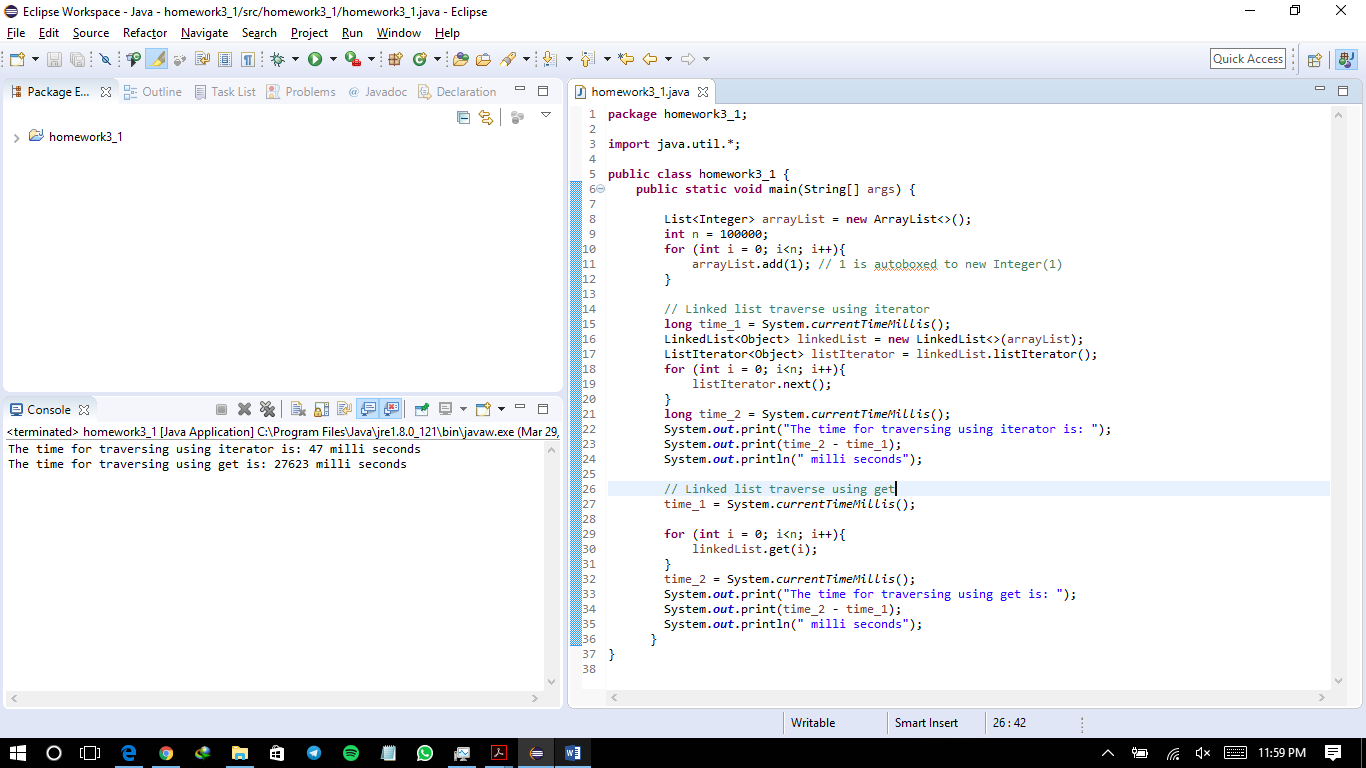
What this question is asking is about how long it takes to print all data of a linked list if we use iterator versus get. If I use iterator it starts from the first element and iterates to the end and prints all of that one by one. However, every single time I trigger get, it starts from the first element and iterates all the way to the target and only prints the target. In other words, as the input element of the get gets bigger, it takes more time to print all data.

The Following simulation is for 100,000 data since it took very long to simulate for 5,000,000 data.

Question 1 Output:

he time for traversing using iterator is: 47 milli seconds

The time for traversing using get is: 27623 milli seconds



Question 2:

**package** homework3\_2;

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

**public** **class** homework3\_2 {

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

PriorityQueue<String> queue1 = **new** PriorityQueue<>();

queue1.offer("George");

queue1.offer("Jim");

queue1.offer("John");

queue1.offer("Blake");

queue1.offer("Kevin");

queue1.offer("Michael");

PriorityQueue<String> queue2 = **new** PriorityQueue<>();

queue2.offer("George");

queue2.offer("Katie");

queue2.offer("Kevin");

queue2.offer("Michelle");

queue2.offer("Ryan");

PriorityQueue<String> Union = **new** PriorityQueue<>();

PriorityQueue<String> Difference = **new** PriorityQueue<>();

PriorityQueue<String> Intersection = **new** PriorityQueue<>();

String A = "";

String B = "";

// While loop until one queue is ended

**while** (queue1.size() > 0 && queue2.size() > 0 ) {

A = queue1.peek();

B = queue2.peek();

//System.out.print(A + " ");

//System.out.println(B);

**if**( A.compareTo(B) == 0){

Intersection.offer(A);

Union.offer(A);

queue1.remove();

queue2.remove();

} **else** **if** (A.compareTo(B) < 0){

Union.offer(A);

Difference.offer(A);

queue1.remove();

} **else** {

Union.offer(B);

Difference.offer(B);

queue2.remove();

}

}

// While for the loop with remaining data

**while** (queue1.size() > 0){

A = queue1.remove();

Union.offer(A);

Difference.offer(A);

}

// While for the loop with remaining data

**while** (queue2.size() > 0){

B = queue2.remove();

Union.offer(B);

Difference.offer(B);

}

System.***out***.print("List of Unions: ");

**while** (Union.size() > 0) {

System.***out***.print(Union.remove() + " ");

}

System.***out***.println("");

System.***out***.print("List of Differences: ");

**while** (Difference.size() > 0) {

System.***out***.print(Difference.remove() + " ");

}

System.***out***.println("");

System.***out***.print("List of Intersections: ");

**while** (Intersection.size() > 0) {

System.***out***.print(Intersection.remove() + " ");

}

System.***out***.println("");

}

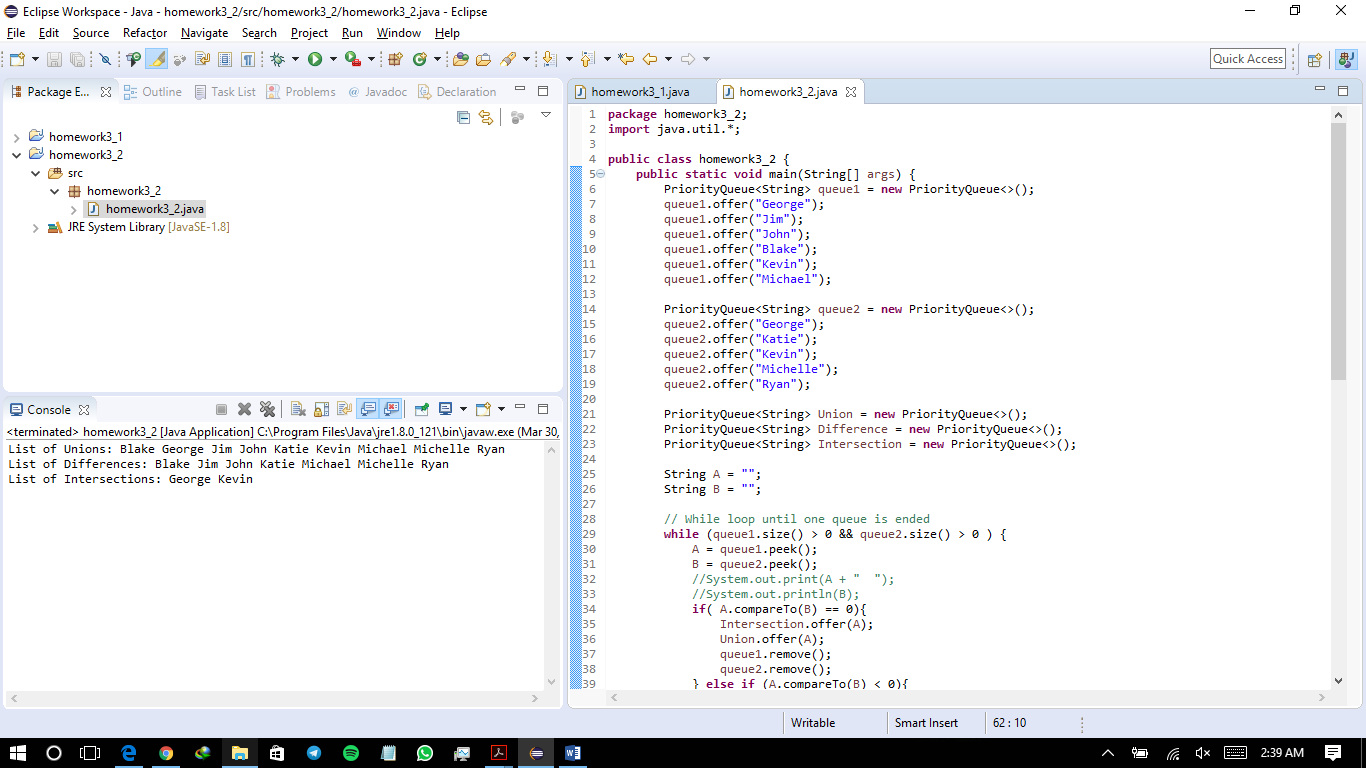
}

Question 2 Output:

List of Unions: Blake George Jim John Katie Kevin Michael Michelle Ryan

List of Differences: Blake Jim John Katie Michael Michelle Ryan

List of Intersections: George Kevin



Question 3:

**package** homework3\_3;

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

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

**public** **class** homework3\_3 {

**public** **static** **void** main(String[] args) **throws** Exception {

Scanner input = **new** Scanner(System.***in***);

System.***out***.print("Enter a Java source file: ");

String filename = input.nextLine();

// C:\Users\ARSH\Box Sync\Personal\Eclipse Workspace\homework3\_3\bin\homework3\_3\text.txt

File file = **new** File(filename);

**if** (file.exists()) {

//System.out.println("The number of keywords in " + filename + " is " + countKeywords(file));

**int** [] result = *countKeywords*(file);

System.***out***.println("The number of vowels and consonants in the file are " + result[0] + " and "+ result[1] );

}

**else** {

System.***out***.println("File " + filename + " does not exist");

}

}

**public** **static** **int**[] countKeywords(File file) **throws** Exception {

String[] keywordString = {"A", "E", "I", "O", "U", "a", "e", "i", "o", "u" };

Set<String> keywordSet = **new** HashSet<>(Arrays.*asList*(keywordString));

**int** count\_vowel = 0;

**int** count\_consonant = 0;

Scanner input = **new** Scanner(file);

System.***out***.println("The string in the file are:");

**while** (input.hasNext()) {

String word = input.next();

System.***out***.println(word + " ");

String [] new\_word = word.split("");

**for** (**int** i = 0; i<word.length();i++){

**if** (keywordSet.contains(new\_word[i])){

count\_vowel++;

}

**else** {

count\_consonant++;

}

}

}

**int**[] result = {count\_vowel, count\_consonant};

**return** result;

}

}

Question 3 Output:

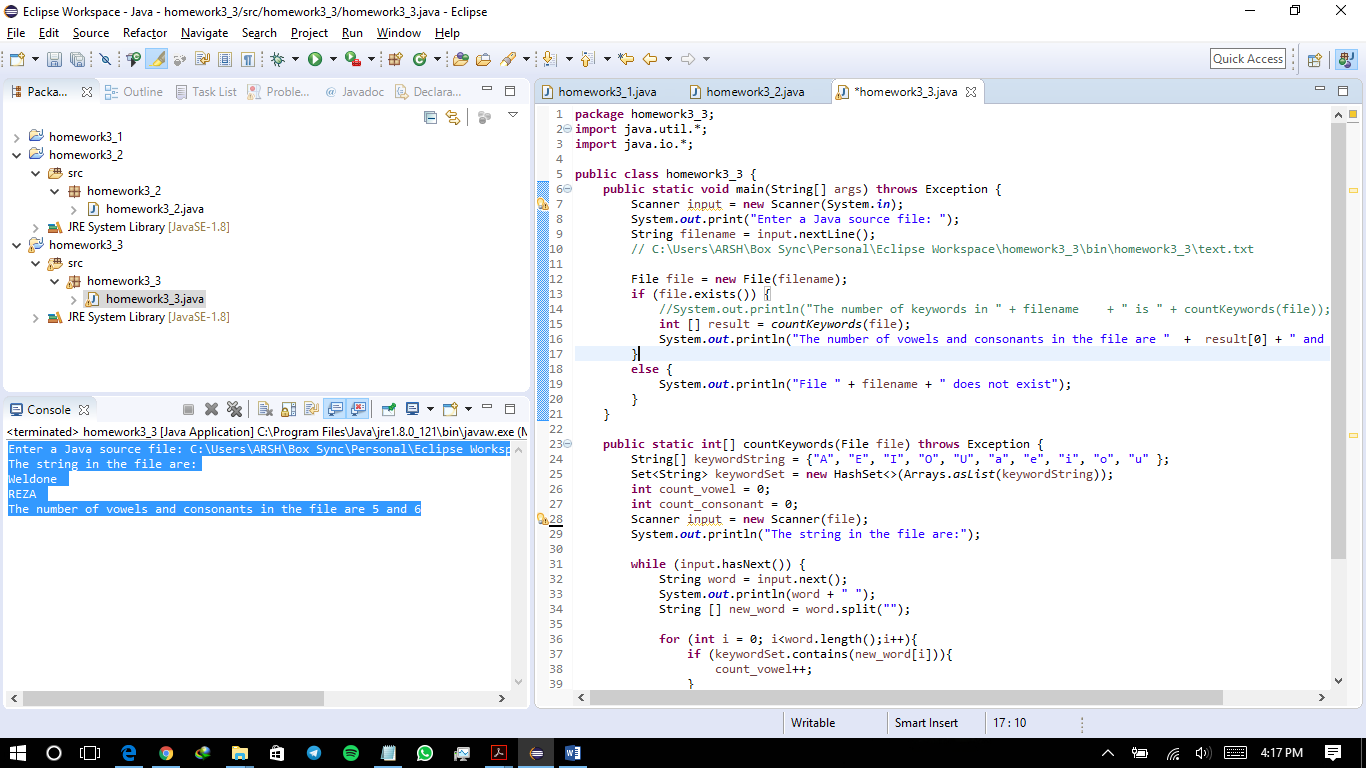
Enter a Java source file: C:\Users\ARSH\Box Sync\Personal\Eclipse Workspace\homework3\_3\bin\homework3\_3\text.txt

The string in the file are:

Weldone

REZA

The number of vowels and consonants in the file are 5 and 6



Question 4:

**package** homework3\_4;

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

**public** **class** homework3\_4 {

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

Map<String, String> linkedHashMap = **new** LinkedHashMap<>(4, 0.75f, **true**);

linkedHashMap.put("Ohio", "Columbus");

linkedHashMap.put("West Virginia", "Charleston");

linkedHashMap.put("California", "Sacramento");

linkedHashMap.put("Texas", "San Antonio");

Set set = linkedHashMap.entrySet();

Iterator i = set.iterator();

Scanner input = **new** Scanner(System.***in***);

**int** correct = 0;

**int** notcorrect = 0;

**while**(i.hasNext()) {

Map.Entry me = (Map.Entry)i.next();

String State = (String)me.getKey();

String Capital = (String)me.getValue();

System.***out***.println("Enter the capital of " + State +":" );

String new\_capital = input.nextLine();

**if** (Capital.equalsIgnoreCase(new\_capital)){

System.***out***.print("Bravo! " );

correct++;

} **else** {

System.***out***.print("No Correct! " );

notcorrect++;

}

System.***out***.println("The capital of " + State +" is " + Capital );

}

System.***out***.println("The number of correct answers are: " + correct);

System.***out***.println("The number of false answers are: " + notcorrect);

}

}

Question 4 Output:

Enter the capital of Ohio:

columbus

Bravo! The capital of Ohio is Columbus

Enter the capital of West Virginia:

CHARLESTON

Bravo! The capital of West Virginia is Charleston

Enter the capital of California:

SaCRAmenTO

Bravo! The capital of California is Sacramento

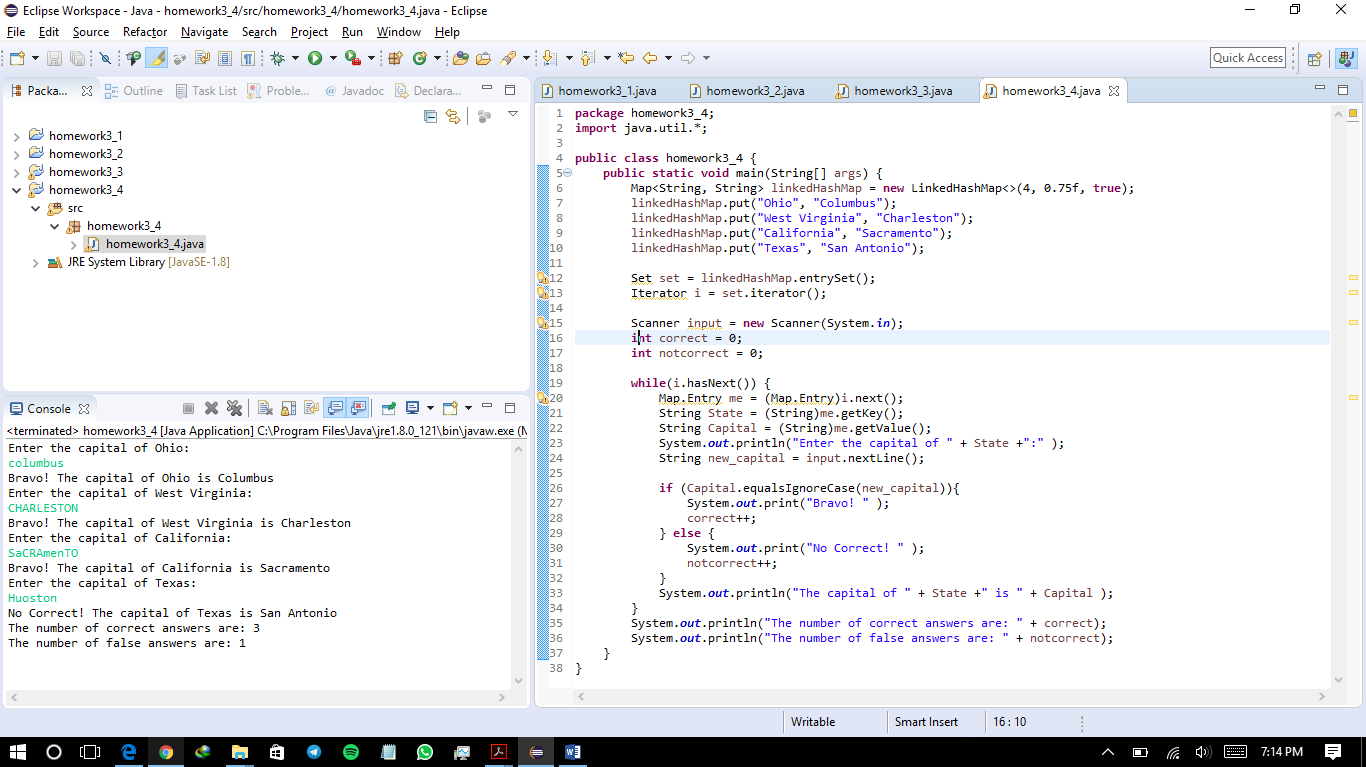
Enter the capital of Texas:

Huoston

No Correct! The capital of Texas is San Antonio

The number of correct answers are: 3

The number of false answers are: 1



Question 5:

**5.1. Greater Common Divisor (GCD) using Euclidean Method:**

What is essential about the Euclid method is that it works recursively and invokes itself until it hits 1. Every time it takes the GCD of n and the remainder of “m%n” which happens “k” times.

The remainder of two number can’t be greater than the half of the largest one which can be show as follows

10%1 = 0

10%2 = 0

10%3 = 1

10%4 = 2

10%5 = 0

10%6 = 4

10%7 = 3

10%8a = 2

10%9 = 1

which are all leass than n / 2 = 5.

Thus every time the “m%n” is invoked, the value gets halved.

gcd(m, n)

gcd(n, m % n)

gcd(m % n, n % (m % n))

as shown above every two iteration the first value gets halved thus the value of “k” should be , which is

Thus the time complexity is O(log *n*)

**5.2. prime number using Sieve of Eratosthenes:**

The Sieve of Eratosthenes’s method eliminates the products of all numbers with itself in an increasing order to the end and counts the remaining unaffected number as prime numbers. If the number is n, the method gets invoked n/2 times for 2, n/3 times for 3 and so on, thus the complexity is:

O (n/2 + n/3 + n/5 + n/11 + … ) = n \* O (1/2 + 1/3 + 1/5 + 1/11 + … ) ≤ O(nF(n)) =