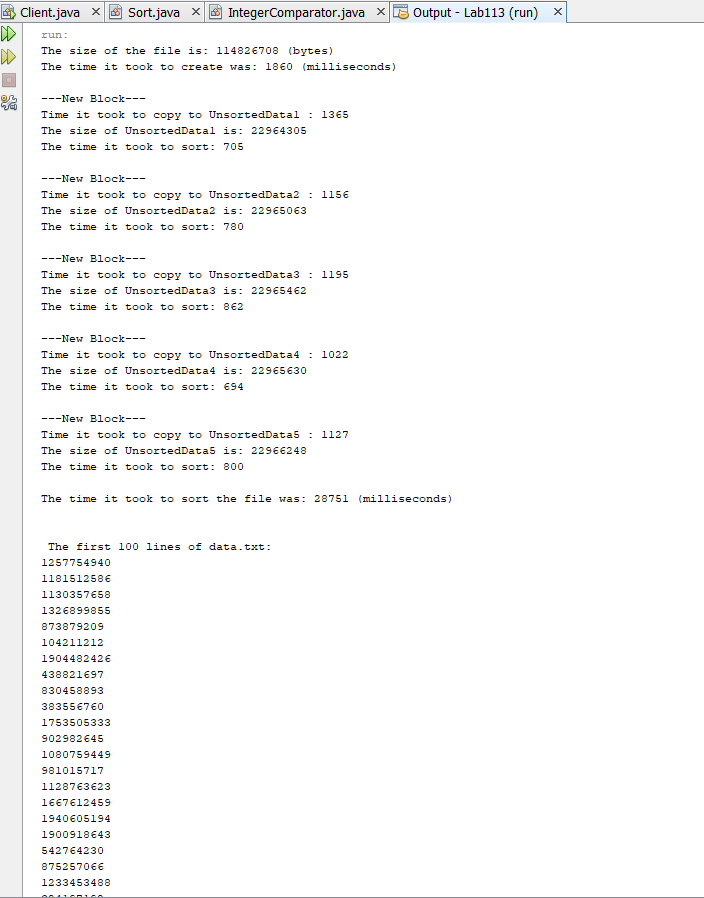
Dylan T Carlson

1161653

Lab112

11/30/18



**Client Class:**

/\*\*

\* This Client class will sort a file of the amount of elements specified

\* by the first instance variable in the main method. To write to a file,

\* Java Illuminated Chapter 11 was used for reference.

\* @author dylca

\*/

import java.io.File;

import java.io.FileOutputStream;

import java.io.PrintWriter;

import java.io.FileNotFoundException;

import java.util.Random;

import java.util.Scanner;

public class Client {

/\* Demonstrating how to write basic data types to a text file.

Anderson, Franceschi

\*/

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

int numberOfElements = 10000000;

long startTime = System.currentTimeMillis();

File data = new File("C:\\data\\data.txt");

File sortedData = new File("C:\\data\\SortedData.txt");

try {

FileOutputStream fos = new FileOutputStream("C:\\data\\data.txt", false);

//false means we will be writing to data.txt

Random rand = new Random();

PrintWriter pw = new PrintWriter( fos );

for(int i = 0; i < numberOfElements; i++){

pw.println( rand.nextInt(Integer.MAX\_VALUE) );

}

//release the reasources associated with data.txt

pw.close();

}

catch( FileNotFoundException fnfe) {

System.out.println("Unable to find data.txt");

}

long endTime = System.currentTimeMillis();

long elasped = endTime - startTime;

System.out.println("The size of the file is: " + data.length() + " (bytes)" );

System.out.println("The time it took to create was: " + elasped + " (milliseconds)" );

startTime = System.currentTimeMillis();

sortDataFile(data, numberOfElements);

endTime = System.currentTimeMillis();

elasped = endTime - startTime;

System.out.println("\nThe time it took to sort the file was: " + elasped + " (milliseconds)" );

System.out.println("\n\n The first 100 lines of data.txt: ");

Scanner origData = new Scanner(data);

Scanner scanSorted = new Scanner(sortedData);

for(int r = 0; r < 100; r++){

System.out.println( origData.nextLine() );

}

System.out.println("\n\n The first 100 lines of SortedData.txt: ");

for(int r = 0; r < 100; r++){

System.out.println( scanSorted.nextLine() );

}

}

/\*\*

\* The start of the process to sort the data file, it breaks it down to

\* 5 different blocks.

\* @param data

\* @param length

\* @throws FileNotFoundException

\*/

public static void sortDataFile(File data, int length)throws FileNotFoundException{

IntegerComparator comp = new IntegerComparator();

Scanner scan = new Scanner(System.in);

File inFile = new File("C:\\data\\data.txt");

Scanner dataScanner = new Scanner(inFile);

int size = length; //the number of elements, this will change

int blockSize = size/5;

File[] UnsortedData = new File[5];

UnsortedData[0]= new File("C:\\data\\SortedData1.txt");

UnsortedData[1]= new File("C:\\data\\SortedData2.txt");

UnsortedData[2]= new File("C:\\data\\SortedData3.txt");

UnsortedData[3]= new File("C:\\data\\SortedData4.txt");

UnsortedData[4]= new File("C:\\data\\SortedData5.txt");

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

System.out.println("\n---New Block---");

long startTime = System.currentTimeMillis();

try {

FileOutputStream fos = new FileOutputStream(UnsortedData[i], false);

//false means we will be writing to UnSortedData1.txt

PrintWriter pw = new PrintWriter( fos );

for(int j = 0; j < blockSize; j++){

String inString = dataScanner.nextLine();

pw.println( inString );

}

//release the reasources associated with UnSortedData.txt

pw.close();

}

catch( FileNotFoundException fnfe) {

System.out.println("Unable to find the file");

}

long endTime = System.currentTimeMillis();

long elasped = endTime - startTime;

System.out.println("Time it took to copy to UnsortedData"+ (i+1) + " : " + elasped);

System.out.println("The size of UnsortedData"+ (i+1) + " is: "+ UnsortedData[i].length());

//Sorts all 5 blocks

sortUnsortedFile(UnsortedData[i], blockSize);

}

//Now to merge the sorted data files together

mergeSortedFiles(UnsortedData, size);

}

/\*\*

\* This method sorts the unsorted files (blocks).

\* @param unsorted

\* @param sizeOfBlock

\*/

public static void sortUnsortedFile(File unsorted, int sizeOfBlock){

int blockSize = sizeOfBlock;

//storing data from block into an array so we can use merge sort

try {

Scanner fileScanner = new Scanner(unsorted);

IntegerComparator intComp = new IntegerComparator();

Integer[] data = new Integer[blockSize];

//Move it to an array

for(int i = 0; i < blockSize; i++){

data[i] = fileScanner.nextInt();

}

//sort the array

long startTime = System.currentTimeMillis();

Sort.mergeSort(data, intComp);

long endTime = System.currentTimeMillis();

long elasped = endTime - startTime;

System.out.println("The time it took to sort: "+ elasped);

FileOutputStream fos = new FileOutputStream(unsorted, false);

//false means we will be writing to UnSortedData1.txt

PrintWriter pw = new PrintWriter( fos );

//Write the array to the file

for(int j = 0; j < blockSize; j++){

pw.println( data[j] );

}

//release the reasources associated with UnSortedData.txt

pw.close();

}

catch (FileNotFoundException fnfe){

System.out.println("File not Found");

}

}

/\*\*

\* Finally, this method merges all the blocks together.

\* @param blocks

\* @param length

\*/

public static void mergeSortedFiles(File[] blocks, int length){

int size = length;

int numOfBlocks = 5;

try {

Scanner block1 = new Scanner(blocks[0]);

Scanner block2 = new Scanner(blocks[1]);

Scanner block3 = new Scanner(blocks[2]);

Scanner block4 = new Scanner(blocks[3]);

Scanner block5 = new Scanner(blocks[4]);

IntegerComparator intComp = new IntegerComparator();

int[] intsToCompare = new int[5];

int minimum = 0;

FileOutputStream fos = new FileOutputStream("C:\\data\\SortedData.txt", false);

//false means we will be writing to data.txt

PrintWriter pw = new PrintWriter( fos );

//initial conditions

boolean[] visited = new boolean[5];

visited[0] = true;

visited[1] = true;

visited[2] = true;

visited[3] = true;

visited[4] = true;

//Find the min and write it to the file

for(int i = 0; i < size; i++){

if(visited[0] && block1.hasNext())

intsToCompare[0] = Integer.parseInt(block1.nextLine());

if(visited[1] && block2.hasNext())

intsToCompare[1] = Integer.parseInt(block2.nextLine());

if(visited[2] && block3.hasNext())

intsToCompare[2] = Integer.parseInt(block3.nextLine());

if(visited[3] && block4.hasNext())

intsToCompare[3] = Integer.parseInt(block4.nextLine());

if(visited[4] && block5.hasNext())

intsToCompare[4] = Integer.parseInt(block5.nextLine());

visited[0] = false;

visited[1] = false;

visited[2] = false;

visited[3] = false;

visited[4] = false;

//Setting min

minimum = findMin(intsToCompare);

//The one that was added must be set to true

for(int h = 0; h < numOfBlocks; h++){

if(intsToCompare[h] == minimum){

visited[h] = true;

//If it does not have next, then is has been used up. So it must not be used.

if(!block1.hasNext())

intsToCompare[0] = -1;

if(!block2.hasNext())

intsToCompare[1] = -1;

if(!block3.hasNext())

intsToCompare[2] = -1;

if(!block4.hasNext())

intsToCompare[3] = -1;

if(!block5.hasNext())

intsToCompare[4] = -1;

break;

}

}

pw.println( minimum );

}

//release the reasources associated with data.txt

pw.close();

}

catch( FileNotFoundException fnfe) {

System.out.println("Unable to find data.txt");

}

}

/\*\*

\* This method is used in mergeUnSortedFiles. It returns the minimum value

\* of the array.

\* @param ints

\* @return

\*/

public static int findMin(int[] ints){

int minimum = Integer.MAX\_VALUE;

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

if(ints[i] < minimum && ints[i] != -1)

minimum = ints[i];

}

return minimum;

}

}

**Int Comparator:**

/\*\*

\* This is a comparator that implements the Comparator interface and

\* compares integers.

\* @author dylca

\*/

public class IntegerComparator implements Comparator<Integer> {

@Override

public int compare(Integer a, Integer b){

if(a > b) return 1;

else if(a < b) return -1;

else return 0;

}

}