Lab104

Steven Glasford

2-19-2019

1 Recursion.java

```java
import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;

/**
 * The Recursion class provides static methods that can calculate
 * a harmonic number, half the size of an array of the power of two, and find
 * a given file within a given path.
 * @author Steven Glasford
 * @version 2-15-2019
 */
public class Recursion {
    /**
     * Given a value, this will determine the given value s harmonic number,
     * using recursion
     * @param given The initial number to determine the harmonic for
     * @return      The harmonic value
     */
    public static double harmonic(int given){
        //base case is if the given number is zero
        if(given < 1){
            return 0;
        }
        //else the recursive statement
        else{
            return 1 / (double) given + harmonic(given - 1);
        }
    }

    /**
     * Sums an array which sinks it down to half its size and stores the summed
     * value for each recursive call
     * @param arr an input array
     * @return  the value of Isabel s technique
     */
    public static int isabel(int[] arr){
        //determine the size of the array given
        int size = arr.length/2;
        //make a temporary array to store the added values together
        int[] temp = new int[size];

        //make a trace of the interaction, this is mostly used in debugging, but
        //still provides an interesting visual to the program
        String trace = "[ ";
        for (int i = 0; i < arr.length; i++){
            trace = trace + arr[i] + ", ";
        }
        trace += " ]";
        System.out.println(trace);

        if (size == 1){
            return arr[0] + arr[1];
        }
        else{
            for (int i = 0; i < size; i++){
                temp[i] = arr[2*i] + arr[2*i+1];
            }
            return isabel(temp);
        }

    }

    /**
     * Find a given file within a given path in the file
     * @param path       The given directory of the file you want to look in
     * @param target     the specific file you are looking for
     */
    public static void find(String path, String target){
        try {
            //open the file from a string
            File root = new File(path);
            //get all of the files within the given directory
            File[] files = root.listFiles();
            //go through every file in search for a directory
            for (File file : files) {
                //if the file is found to be a directory, then recurse
                if (file.isDirectory()) {
                    find(file.getAbsolutePath(), target);
                    //System.out.println(file.getCanonicalPath());
                    //printTree(file);
                } else {
                    //if the file is found then return the full name of the file
                    if (file.getName().equals(target)) {
                        System.out.println(file.getAbsolutePath());
                    }
                }
            }
        }
        //throw errors if the directory is not found or is a file, very general
        //throw statement
        catch(NullPointerException e){

        }
    }


}
```

2 Client.java

```java
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
import javax.swing.JOptionPane;
/**
 * A main client class controlling the bulk of the programming; uses
 * JOptionPane to display information and control the program, and implements
 * Isabel s summation technique, a harmonic number calculator, and a file
 * finding method.
 * @author Steven Glasford
 * @version 2-15-2019
 */
public class Client {

    /**
     * Pop up a window saying an option was not chosen if no option is chosen,
     * sort of a de facto.
     */
    public static void noOptionSelected(){
        JOptionPane.showMessageDialog(null, "Nothing was entered, "
                + "please enter something.");
    }

    /**
     * Ask for a confirmation that the user wants to quit the program
     * @return  true if the user wants to quit, false if the user does not want
     * to quit.
     */
    public static boolean confirmExit(){
        int option = JOptionPane.showConfirmDialog(null,"Are you sure you "
                + "want to exit?","exit", JOptionPane.YES_NO_OPTION);
        return JOptionPane.YES_OPTION == option;
    }

    /**
     * If harmonic calculator selected, prompts user for value and ensure it is
     * safe for use.
     */
    public static void runHarmonic(){
        System.out.println("Testing Harmonic calculator");
        //used to stop the program
        boolean stop = false;

        //used to get the number for the given number to be calculated
        int result = 0;

        String input = JOptionPane.showInputDialog(null,"Please enter in "
                + "a integer number","Harmonic Calculator",
                JOptionPane.OK_CANCEL_OPTION);
        //check if the user inputted something
        if(input == null){
            //ask if the user wants to quit
            if (confirmExit()){
                return;
            }
            runHarmonic();
            return;
        }
        //try casting into an integer, throw an error if not castable
        try{
            result = Integer.parseInt(input);
        }
        //throw an number exception error
        catch (NumberFormatException e){
            //print an error message in the terminal
            System.out.println("Invalid parameter: " + input);
            //produce an error message in the JOptionPane
            JOptionPane.showMessageDialog(null, "Not an integer value, "
                    + "please enter integer");
            //run the program recursively from this point
            runHarmonic();
            //end the method immediately after the program is brought back from
            //the recursive run, eliminating break statement
            return;
        }

        //throw an error message if zero is entered
        if (result == 0){
            //deliver an error in the terminal
            System.out.println("Zero was entered.");
            //give an error in the dialog
            JOptionPane.showMessageDialog(null, "Harmonic is undefined "
                + "at 0, please enter something else.");
            //run the program recursively otherwise
            runHarmonic();
            //end the method after returning from the recursive statement
            return;
        }
        //produce an error if the number is less than zero
        if (result < 1){
            //print an error statement to the terminal
            System.out.println("User entered a negative number.");
            //print an error statement to the dialog
            JOptionPane.showMessageDialog(null, "Not a positive number."
                    + " enter something else.");
            //run the program recursively
            runHarmonic();
            //end the method once the recursive statement is finished.
            return;
        }
        //print a confirmation that the parameter entered was legal to the
        //terminal
        System.out.println("Valid parameter: " + result);
        //save the value of the harmonic value
        double harmonic = Recursion.harmonic(result);
        //print the harmonic value to the terminal
        System.out.println("Output: "+ harmonic);
        //print the harmonic value to the dialog
        JOptionPane.showMessageDialog(null, Double.toString(harmonic));
        //end the method if the method and return to the other screen
        return;
}
   /**
   * Checks to ensure a number is the power of two
   * @param number    any number to test
   * @return  true if the number is a power of two, false otherwise.
   */
```

```java
public static boolean isPowerOfTwo(int number) {

    // a number cannot be a power of two if it is not even
    if (number % 2 != 0)
        return false;

    else {
        for (int i = 0; i <= number; i++) {
            //return true if the number is a power of two
            if (Math.pow(2, i) == number)
                return true;
        }
    }

    //return false elsewise
    return false;
}


/**
 * A method to appropriately run Isabel s method.
 */
public static void runIsabel(){
    //print out a title page
    System.out.println("Isabel sum");
    //the string to the path a person wants to take
    String path;
    //a new array of the ArrayBag format
    ArrayBag fromFile;
    fromFile = new ArrayBag() {};

    path = JOptionPane.showInputDialog(null,"Please enter file path");
    if(path == null){
        if (confirmExit()){
            return;
        }
        //produce an error message to the terminal
        System.out.println("Nothing was entered.");
        //produces an error message in the dialog box
        JOptionPane.showMessageDialog(null, "Nothing entered,"
            + " please enter something.");
        //recurse the method
        runIsabel();
        //kill the method once finished
        return;
    }

    //make a file instance
    File file;
    //start a scanner instance that will look into the given file
    Scanner scan = null;

    //try to open the given file
    try {
        file = new File(path);
        scan = new Scanner(file).useDelimiter(" ");
        System.out.println("Valid path: " + path);

    }
    //throws a file not found exception
    catch(FileNotFoundException e){
        //produce an error message in the terminal
        System.out.println("Invalid path: " + path);
        //produce an error message in the dialog box
        JOptionPane.showMessageDialog(null, "Not a valid file location, "
                + "please enter valid path");
        //recursively run the method
        runIsabel();
        //kill the method once the recursed method is finished.
        return;
    }

    //read the information from the file into an array
    try{
        while(scan.hasNext()){
            //if the information is not an integer format, then skip over
            //that piece of information
            try {
                fromFile.add(Integer.parseInt(scan.next()));
            }
            catch (NumberFormatException e){
            }
        }
    }
    //generic error thrown if the file cannot be read for whatever reason
    catch(NullPointerException e){    }

    if(!(isPowerOfTwo(fromFile.getCurrentSize()))){
        //produce an error message in the terminal
        System.out.println("Array not a power of two: "
                + fromFile.getCurrentSize());
        //produce an error message in the dialog box
        JOptionPane.showMessageDialog(null, "Array is not a power of two,"
                + " please enter valid path of int array that contains "
                + "length that is power of two");
        //recurse the program
        runIsabel();
        //kill the method after returning from the recursed method
        return;
    }

    //
    System.out.println("input: "+ fromFile.toString());

    //produce the output of the Isabel s method to the terminal
    System.out.println("Output: " +
            Recursion.isabel(fromFile.getIntArray()));

    //kill the method if everything is good in the world.
    return;


}

public static void runFind() throws FileNotFoundException{
    System.out.println("Find file");


    String path;
    path = JOptionPane.showInputDialog(null,"Please enter file path");
```

```
        String target = JOptionPane.showInputDialog(null, "Please enter the "
                + "target file.");
        if(null == path){
            //determine if the user wants to continue with their option or not
            if (confirmExit()){
                //kill the method if they don t want to continue
                return;
            }
            //recursively run the program if the user wants to continue
            runFind();
            //kill the method if they want to quit
            return;
        }

        //open the file, regardless of whether it exists or any of that
        //shit
        File file = new File(path);

        //all you care about is if it is a directory or not
        if(!(file.isDirectory())){
            //print an error message to the terminal
            System.out.println("Invalid path: " + path);
            //print an error message to the dialog
            JOptionPane.showMessageDialog(null, "This is not a valid,"
                    + " file please enter valid path to directory");
            //recursively run the program
            runFind();
            //kill the method if after the recursion
            return;
        }

        //print the path of inputted path to the terminal
        System.out.println("Path: " + path);

        //search for the file
        Recursion.find(path,target);

        //kill the method once finished
        return;
    }

   /**
    * The menu system that prompts user to select an operation or exit the
    * program
     * @throws java.io.FileNotFoundException
    */
    public static void menuSelector() throws FileNotFoundException{
        String optionString;
        optionString = "A to run harmonic calculator"
                + "\nB to run Isabel s sum"
                + "\nC to run find file"
                + "\nD to exit";
        String response = JOptionPane.showInputDialog(null,optionString);
        if(null == response)
            if(confirmExit())
                return;

        switch ( response ){
            case "a" :
            case "A" :
                runHarmonic();
                //recurse the menuSelector
                menuSelector();
                break;
            case "b" :
            case "B" :
                runIsabel();
                //recurse the menuSelector
                menuSelector();
                break;
            case "C":
            case "c":
                runFind();
                //recurse the menu
                menuSelector();
                break;
            case "D":
            case "d":
                //confirm if the user wants to quit
                confirmExit();
                break;
            default :
                //tell the user that nothing was entered
                noOptionSelected();
                menuSelector();
                break;
        }
    }

    /**
     * Starts the menu system and the menu system will end itself.
     * @param args the command line arguments
     * @throws java.io.FileNotFoundException
     */
    public static void main(String[] args) throws FileNotFoundException {
        menuSelector();
    }

}
```

3 ArrayBag.java

```java
import java.util.Arrays;

/**
 * A class that uses the ability to make an array bag
 * @author Steven Glasford
 * @version 2-15-2019
 *
 * @param <A>
 */
public abstract class ArrayBag<A> implements Bag{
    A[] list;
    public int count;

    /**
     * A default constructor the generic array for the class
     */
    public ArrayBag () {
        list = (A[]) new Object[50];
    }

    /**
     * A non-default constructor for the array bag
     * @param size
     */
    public ArrayBag (int size){
        if (size <= 0){
            size = 50;
        }

        list = (A[]) new Object[size];

    }

    /**
     * Used to get the size of the bag
     * @return The current size of the bag
     *
     */
    @Override public int getCurrentSize(){
        return count;
    }

    /**
     * Determine if the bag is empty
     * @return whether or not the bag is empty
     */
    @Override public boolean isEmpty(){
        return (count == 0);
    }

    /**
     * Used to clear and destroy the bag the bag
     */
    @Override public void clear( ) {
        count = 0;
    }

    /**
     * Add to the list a number
     * @param thing - add a given number to the bag
     *
     */
    @Override public void add (Object thing){
        //add to the count of the bag first off
        count++;
        A[] temp;

        //double the size of the array if the array is full
        if (count >= list.length){
            //create a new array
            temp = (A[]) new Object[(list.length * 2)];

            //copy the old array into the new one
            for (int i = 0; i < list.length; i++)
                temp[i] = list[i];

            //assign the reference of temp to list
            list = temp;

            //"delete" the temporary array
            temp = null;
        }

        //add the number to the list
        list[count- 1] = (A) thing;
    }

    /**
     * Remove a given item from the bag
     * @param thing
     * @return Return whether or not the item was removed from the bag
     */
    @Override public boolean remove (Object thing){
        int i = 0;

        //try to find the given number
        while (i < count){
            //if the number is found we will remove the first found instance
            //and reduce the size of the bag
            if (list[i].equals(thing)){

                //move the numbers down one from the point of the found number
                for (int j = i; j < count; j++)
                    list[j] = list[j+1];

                //reduce the count of the numbers
                count -= 1;

                //stop the loop without using a break because that is habit
                return true;
            }
            //go to the nuext number in the list
            i++;
        }
        return false;
    }

    /**
```

```
 * Randomly remove a number in the bag
 * @return the item that was removed
 */
@Override public A remove( ){
    //get a random number to remove between 0 and the count - 1
    int random = (int)(Math.random() * count + 1);
    //get the thing that is going to be removed
    A temp = list[random - 1];

    //remove the randomly generated number from the bag
    for (int i = random; i < count; i++){
        list[i-1] = list[i];
    }

    //reduce the size of the bag
    count -= 1;

    //return the found item
    return temp;
}

/**
 * Find the frequency of a given number in the list
 * @param thing Search for thing in the bag and return the frequency of it
 * @return the frequency of the given number in the bag
 */
@Override public int getFrequencyOf(Object thing){
    int temp = 0;

    for (int i = 0; i < count; i++)
        if (list[i].equals(thing))
            temp++;

    return temp;
}

/**
 * Search through the bag to see if the given number is inside the bag
 * @param thing determine if the thing is found in the bag
 * @return true or false depending on if the bag contains a given number
 */
@Override public boolean contains(Object thing){
    //a loop to go through every instance of the bag to find the given number
    for (int i = 0; i < count; i++)
        if (list[i].equals(thing))
            //return true if found
            return true;


    //return false if not found
    return false;
}

/**
 * Convert the array into a string
 * @return the bag as a string
 */
@Override public String toString( ){
    //create the string
    String words;
```

```
        words = "[ ";

        //build the string
        for (int i = 0; i < count; i++){
            words = words + list[i] + ", ";
        }

        words += " ]";

        return words;
    }

    /**
     * Netbeans suggested I put this in here
     * @return true or false depending on if the object is identical to the bag
     */
    @Override public int hashCode(){
        int hash = 3;
        hash = 79 * hash + Arrays.deepHashCode(this.list);
        hash = 79 * hash + this.count;
        return hash;
    }


    /**
     * Determine if the two bags are equal
     * @param o a generic object to test the equality
     * @return whether the objects are equal
     */
    @Override
    public boolean equals(Object o) {
        //first test to see if the object is an int array
        if (!(o instanceof int[])){
            return false;
        }

        //check if the object is the same size as list[]
        A[] b;
        b = (A[]) o;
        if (!(b.length == list.length))
            return false;

        //go through the object and test for equality with respect to list[]
        for (int i = 0; i < count; i++)
            if (b[i] != list[i])
                return false;

        return true;
    }

    /**
     * Get the number stored in i-th position in the bag
     * @param i - a place in the bag array
     * @return The value stored in the array at the position, not in the bag
     */
    public A get(int i) {
        A info = null;

        try {
            info = list[i];
        }
        catch (ArrayIndexOutOfBoundsException exception){
            System.out.println(exception);
        }

        return info;
    }

    /**
     * Convert the array into an int array, this is only used if the
     * coder knows what they are doing, as not all bags are convertable to int
     * @return The bag as an int array
     */
    public int[] getIntArray(){
        int[] temp;
        temp = new int[count];
        for (int i = 0; i < count; i++){
            temp[i] = (int) list[i];
        }
        return temp;
    }
}
```