START Moon Class

//instance variables

private String name;

private double radiusl;

private double density;

private double distance;

//default constructor

public Moon(){

name = “”;

radius = 0.0;

density = 0.0;

distance = 0.0;

}

//parameterized constructor

public Moon(String newName, double newRadius, double newDensity, double newDistance){

name = newName;

radius = newRadius;

density = newDensity;

distance = newDistance;

}

public String getName(){

return name;

}

public double getRadius(){

return radius;

}

public double getDensity(){

return density;

}

public double getDistance(){

return double;

}

//do not forget @Override

public String toString(){

String returnRadius = String.format("%.2f", radius);

String returnDensity = String.format("%.2f", density);

String returnDistance = String.format("%.2f", distance);

return name + “ “ + returnRadius + “ “ + returnDensity + “ “ + returnDistance;

}

END Moon Class

START ENUMERATED TYPE MoonAttributes

public enum MoonAttributes {

RADIUS, DENSITY, DISTANCE

}

END ENUMERATED TYPE

START openFile() 🡪 public static ArrayList<String> openFile(File inputFile) throws FileNotFoundException

Scanner input = new Scanner(inputFile);

ArrayList<String> stringArray = new ArrayList<String>();

while (input.hasNextLine()){

//adds the line as a single String element in the String ArrayList

stringArray += input.nextLine();

}

return stringArray;

END openFile()

START createObjects() 🡪 public static ArrayList<Moon> createObjects(ArrayList<String> lines)

//everything beneath this for loop needs to go inside the for loop for this method to work

//need to figure out what to replace “IS INCODRRECT DATA” with

START FOR (int index = 0; index < lines.size(); i++)

String[] tempArray = lines.get(i).splits(“\t”);

ArrayList<Moon> moonList = new ArrayList<Moon>();

//TEST IF SECOND ELEMENT IS VALID RADIUS

try {

double newRadius = Double.parseDouble(tempArray[1]);

}

catch (NumberFormatException exception){

if (Double.parseDouble(tempArray[1]) < 0)

double newRadius = 0.0;

if (Double.parseDouble(tempArray[1]) IS INCORRECT DATA)

double newRadius = 0.0;

}

//TEST IF THIRD ELEMENT IS A VALID DENSITY

try {

double newDensity = Double.parseDouble(tempArray[2]);

}

catch (NumberFormatException exception){

if (Double.parseDouble(tempArray[2]) < 0)

double newDensity = 0.0;

if (Double.parseDouble(tempArray[2]) IS INCORRECT DATA)

double newDensity = 0.0;

}

//TEST IS FOURTH ELEMENT IS A VALID DISTANCE

try {

double newRadius = Double.parseDouble(tempArray[3]);

}

catch (NumberFormatException exception){

if (Double.parseDouble(tempArray[3]) < 0)

double newDistance = 0.0;

if (Double.parseDouble(tempArray[3]) IS INCORRECT DATA)

double newDistance = 0.0;

}

//once we have created the new variables, we can create a new Moon obj and add it

//to the moonList Moon ArrayList

Moon moonObj = new Moon(tempArray[0], newRadius, newDensity, newDistance);

moonList.add(moonObj);

END FOR

return moonList;

END createObjects()

START findMean() 🡪 public static double findMean(ArrayList<Moon> moons,

MoonAttributes attribute)

double average = 0;

int count = 0;

/\*

create if statement to see which attribute we will be assessing. Then

within each if statement we have a for loop to grab the attribute of

each Moon object in the moons ArrayList

\*/

if (attributes == MoonAttributes.RADIUS){

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

average += moons.get(i).getRadius();

count++;

}

}

else if (attributes == MoonAttributes.DENSITY){

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

average += moons.get(i).getDensity();

count++;

}

}

else if (attributes == MoonAttributes.DISTANCE){

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

average += moons.get(i).getDistance();

count++;

}

}

average = average / count;

return average;

END findMean()

START findHighValue() 🡪 public static double findHighValue(ArrayList<Moon>

moons, MoonAttributes attribute){

/\*

If statements set up to see which attribute we are testing. Then inside there is a for loop that checks for the highest max

\*/

double max = 0.0;

if (attribute == MoonAttributes.RADIUS){

max = moons.get(0).getRadius();

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

if (moons.(i).getRadius() > max){

max = moons.(i).getRadius();

}

}

}

else if (attribute == MoonAttributes.DENSITY){

max = moons.get(0).getDensity();

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

if (moons.(i).getDensity() > max){

max = moons.(i).getDensity();

}

}

}

else if (attribute == MoonAttributes.DISTANCE){

max = moons.get(0).getDistance();

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

if (moons.(i).getDistance() > max){

max = moons.(i).getDistance();

}

}

return max;

}

END findHighValue()

START findMeanMoon() 🡪 public static Moon findMeanMoon(ArrayList<Moon>

moons, MoonAttributes attribute, double meanValue)

* One possible solution: order all elements of the arraylist’s attributes in ascending order (include meanValue in this ordering) and look at number to meanValue’s immediate right and left. Then test which is closer to meanValue

ArrayList<Double> tempArray = new ArrayList<Double>();

tempArray.add(meanValue);

double closestValue = tempArray.get(0);

if (attribute == MoonAttributes.RADIUS){

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

tempArray.add(moons.get(i).getRadius())

}

}

else if (attribute == MoonAttributes.DENSITY){

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

tempArray.add(moons.get(i).getDensity())

}

}

else if (attribute == MoonAttributes.DISTANCE){

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

tempArray.add(moons.get(i).getDistance())

}

}

//this calls the Array’s sort method, which will sort the entire array into ascending order

tempArray = Arrays.sort(0, tempArray.size());

//now we will find the location of the meanValue

int meanValueLocation = 0;

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

if (meanValue == tempArray.get(i)){

meanValueLocation = tempArray.get(0);

}

}

//now we will compare the number to left and right and see which is closer to meanValue

double leftNumber = tempArray.get(0);

if (tempArray.get(meanValueLocation - 1) >= 0){

leftNumber = tempArray.get(meanValueLocation – 1);

}

double rightNumber = 0.0;

if (tempArray.get(meanValueLocation +1) < tempArray.size()){

rightNumber = tempArray.get(meanValueLocation +1);

}

if ((rightNumber – meanValue) < (meanValue – leftNumber)){

closestValue = rightNumber;

}

else {

closestValue = leftNumber;

}

return closestValue;

END findMeanMoon()

START findLowestMoons() 🡪 public static ArrayList<Moon> findLowestMoons(ArrayList<Moon>

moons, double value, MoonAttributes attribute){

ArrayList<Moon> leastMoons = new ArrayList<Moon>();

if (attribute = MoonAttributes.RADIUS){

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

if (moons.get(i).getRadius() < value){

leastMoons.add(moons.get(i));

}

}

}

else if (attribute = MoonAttributes.DENSITY){

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

if (moons.get(i).getDensity() < value){

leastMoons.add(moons.get(i));

}

}

}

else if (attribute = MoonAttributes.DISTANCE){

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

if (moons.get(i).getDistance() < value){

leastMoons.add(moons.get(i));

}

}

}

return leastMoons;

END findLowestMoons()

outputToFile method for ARRAY LIST OF VALUES 🡪 public static void outputToFile(String outputMessage,

ArrayList<Moon> moons, PrintWriter out)

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

outputMessage += moons.get(i).toString();

outputMessage += “\n” + “\n” // to have new line and then a blank line to seperate

}

out.print(outputMessage);

out.close();

END

outputToFile method for ONE MOON OBJECT 🡪 public static void outputToFile(String outputMessage,

Moon moon, PrintWriter out)

out.print(moon.toString());

out.close();

END

outputToFile method for PRINTING A DOUBLE VALUE 🡪 public static void outputToFile(String outputMessage, double value, PrintWriter out)

String formattedDouble = String.format("%.2f", value);

out.print(formattedDouble)

END

START MAIN METHOD

\*\*\*\*Accept input and output file from command line\*\*\*\*

Place openFile method in try/catch block

* PRINT “Incorrect input filename” if there is a problem
* PRINT “Input file correct” if there isn’t any problems

Call creatObjects() with openFile() String array

* Return value of createObjects is now used for findMean(), findHighValue(), findMeanMoon(), and findLowestMoons()

Call writeOutData, and place in try/catch block

* If problem PRINT “Incorrect output filename”
* If no problem PRINT “Output file correct”
* CLOSE the PrintWriter file

END MAIN METHOD