package demos;

import processing.core.PApplet;

import processing.core.PImage;

/\*\*

\* A class to illustrate some use of the PApplet class

\* Used in module 3 of the UC San Diego MOOC Object Oriented Programming in Java

\*

\* @author UC San Diego Intermediate Software Development MOOC team

\*

\*

\*/

public class MyPApplet extends PApplet{

PImage img;

public void setup() {

//Add setup code for MyPApplet

size(400,400); //set canvas size

background(255); //set canvas color

stroke(0); //set pen color

img = loadImage("palmTrees.jpg", "jpg");

img.resize(0, height); //resize loaded image to full height of canvas

image(img, 0, 0); //display image

}

public void draw() {

//Add drawing code for MyPApplet

int[] color = sunColorSec(second()); //calculate color code for sun

fill(color[0],color[1],color[2]); //set sun color

ellipse(width/4,height/5,width/4,height/5); //draw sun

}

/\*\* Return the RGB color of the sun at this number of seconds in the minute \*/

public int[] sunColorSec(float seconds)

{

int[] rgb = new int[3];

// Scale the brightness of the yellow based on the seconds. 0 seconds

// is black. 30 seconds is bright yellow.

float diffFrom30 = Math.abs(30-seconds);

float ratio = diffFrom30/30;

rgb[0] = (int)(255\*ratio);

rgb[1] = (int)(255\*ratio);

rgb[2] = 0;

//System.out.println("R" + rgb[0] + " G" + rgb[1] + " B" + rgb[2]);

return rgb;

}

public static void main (String[] args) {

//Add main method for running as application

PApplet.main(new String[] {"--present", "MyPApplet"});

}

}

**package** demos;

**import** processing.core.PApplet;

/\*\*

\* A class to illustrate some use of the PApplet class

\* Used in module 3 of the UC San Diego MOOC Object Oriented Programming in Java

\*

\* **@author** UC San Diego Intermediate Software Development MOOC team

\*

\*

\*/

**public** **class** MyDisplay **extends** PApplet {

**public** **void** setup()

{

size(400, 400);

background(200, 200, 200);

}

**public** **void** draw()

{

fill(255, 255, 0);

ellipse(200, 200, 390, 390);

fill(0, 0, 0);

ellipse(120, 130, 50, 70);

ellipse(280, 130, 50, 70);

noFill();

arc(200, 280, 75, 75, 0, ***PI***);

}

}

**package** demos;

/\*\*

\* A class to illustrate class design and use.

\* Used in module 2 of the UC San Diego MOOC Object Oriented Programming in Java

\*

\* **@author** UC San Diego Intermediate Software Development MOOC team

\*

\*

\*/

**public** **class** SimpleLocation

{

**public** **double** latitude;

**public** **double** longitude;

**public** SimpleLocation()

{

**this**.latitude = 32.9;

**this**.longitude = -117.2;

}

**public** SimpleLocation(**double** latIn, **double** lonIn)

{

**this**.latitude = latIn;

**this**.longitude = lonIn;

}

// Returns the distance in km between this SimpleLocation and the

// parameter other

**public** **double** distance(SimpleLocation other)

{

**return** getDist(**this**.latitude, **this**.longitude,

other.latitude, other.longitude);

}

**private** **double** getDist(**double** lat1, **double** lon1, **double** lat2, **double** lon2)

{

**int** R = 6373; // radius of the earth in kilometres

**double** lat1rad = Math.*toRadians*(lat1);

**double** lat2rad = Math.*toRadians*(lat2);

**double** deltaLat = Math.*toRadians*(lat2-lat1);

**double** deltaLon = Math.*toRadians*(lon2-lon1);

**double** a = Math.*sin*(deltaLat/2) \* Math.*sin*(deltaLat/2) +

Math.*cos*(lat1rad) \* Math.*cos*(lat2rad) \*

Math.*sin*(deltaLon/2) \* Math.*sin*(deltaLon/2);

**double** c = 2 \* Math.*atan2*(Math.*sqrt*(a), Math.*sqrt*(1-a));

**double** d = R \* c;

**return** d;

}

}

**package** demos;

**public** **class** LocationTester {

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

{

SimpleLocation ucsd = **new** SimpleLocation(32.9, -117.2);

SimpleLocation lima = **new** SimpleLocation(-12.0, -77.0);

//latitude = -12.04;

System.***out***.println(ucsd.distance(lima));

}

}

package demos;

import processing.core.PApplet;

import de.fhpotsdam.unfolding.UnfoldingMap;

import de.fhpotsdam.unfolding.utils.MapUtils;

import de.fhpotsdam.unfolding.providers.\*;

import de.fhpotsdam.unfolding.providers.Google.\*;

import java.util.List;

import de.fhpotsdam.unfolding.data.Feature;

import de.fhpotsdam.unfolding.data.GeoJSONReader;

import java.util.HashMap;

import de.fhpotsdam.unfolding.marker.Marker;

/\*\*

\* Visualizes life expectancy in different countries.

\* It loads the country shapes from a GeoJSON file via a data reader, and loads the population density values from

\* another CSV file (provided by the World Bank). The data value is encoded to transparency via a simplistic linear

\* mapping.

\*/

public class LifeExpectancy extends PApplet {

UnfoldingMap map;

HashMap<String, Float> lifeExpMap;

List<Feature> countries;

List<Marker> countryMarkers;

public void setup() {

size(800, 600, OPENGL);

map = new UnfoldingMap(this, 50, 50, 700, 500, new Google.GoogleMapProvider());

MapUtils.createDefaultEventDispatcher(this, map);

// Load lifeExpectancy data

lifeExpMap = loadLifeExpectancyFromCSV("LifeExpectancyWorldBankModule3.csv");

println("Loaded " + lifeExpMap.size() + " data entries");

// Load country polygons and adds them as markers

countries = GeoJSONReader.loadData(this, "countries.geo.json");

countryMarkers = MapUtils.createSimpleMarkers(countries);

map.addMarkers(countryMarkers);

// Country markers are shaded according to life expectancy (only once)

shadeCountries();

}

public void draw() {

// Draw map tiles and country markers

map.draw();

}

//Helper method to color each country based on life expectancy

//Red-orange indicates low (near 40)

//Blue indicates high (near 100)

private void shadeCountries() {

for (Marker marker : countryMarkers) {

// Find data for country of the current marker

String countryId = marker.getId();

if (lifeExpMap.containsKey(countryId)) {

float lifeExp = lifeExpMap.get(countryId);

// Encode value as brightness (values range: 40-90)

int colorLevel = (int) map(lifeExp, 40, 90, 10, 255);

marker.setColor(color(255-colorLevel, 100, colorLevel));

}

else {

marker.setColor(color(150,150,150));

}

}

}

//Helper method to load life expectancy data from file

private HashMap<String, Float> loadLifeExpectancyFromCSV(String fileName) {

HashMap<String, Float> lifeExpMap = new HashMap<String, Float>();

String[] rows = loadStrings(fileName);

for (String row : rows) {

// Reads country name and population density value from CSV row

// NOTE: Splitting on just a comma is not a great idea here, because

// the csv file might have commas in their entries, as this one does.

// We do a smarter thing in ParseFeed, but for simplicity,

// we just use a comma here, and ignore the fact that the first field is split.

String[] columns = row.split(",");

if (columns.length == 6 && !columns[5].equals("..")) {

lifeExpMap.put(columns[4], Float.parseFloat(columns[5]));

}

}

return lifeExpMap;

}

}

**package** demos;

/\*\* A class to store information about a Student

\* Used in module 4 of the UC San Diego MOOC Object Oriented Programming in Java

\*

\* **@author** UC San Diego Intermediate Software Development MOOC team

\*

\*/

**public** **class** Student **extends** Person

{

**public** Student(String name) {

**super**(name);

}

**public** **boolean** isAsleep( **int** hr ) // override

{

**return** 2 < hr && 8 > hr;

}

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

{

Person p;

p = **new** Student("Sally");

p.status(1);

}

}

**package** demos;

/\*\* A class to store information about a Person\*/

**public** **class** Person

{

**private** String name;

**public** Person(String name)

{

**this**.name = name;

}

**public** **boolean** isAsleep(**int** hr)

{

**return** 22 < hr || 7 > hr;

}

**public** String toString()

{

**return** name;

}

**public** **void** status( **int** hr )

{

**if** ( **this**.isAsleep( hr ) )

System.***out***.println( "Now offline: " + **this** );

**else**

System.***out***.println( "Now online: " + **this** );

}

}

package demos;

import java.io.BufferedReader;

import java.io.FileInputStream;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.Collections;

import java.util.Random;

/\*\*

\* A class to illustrate some searching and sorting algorithms

\* Used in module 6 of the UC San Diego MOOC Object Oriented Programming in Java

\*

\* @author UC San Diego Intermediate Software Development MOOC team

\*

\*

\*/

public class SearchAndSort {

// Read the airpoirts in from the file.

private static ArrayList<Airport> readFile(String fname) throws IOException

{

ArrayList<Airport> airports = new ArrayList<Airport>();

FileInputStream fis = new FileInputStream(fname);

BufferedReader br = new BufferedReader(new InputStreamReader(fis));

String line;

while ((line = br.readLine()) != null)

{

String[] data = line.split(",(?=([^\"]\*\"[^\"]\*\")\*[^\"]\*$)");

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

data[i] = data[i].replace("\"", "");

}

int airportID = Integer.parseInt(data[0]);

String name = data[1];

String city = data[2];

String country = data[3];

String code3 = data[4];

String code4 = data[5];

double lat = Double.parseDouble(data[6]);

double lon = Double.parseDouble(data[7]);

int alt = Integer.parseInt(data[8]);

float tz = Float.parseFloat(data[9]);

char dst = data[10].charAt(0);

String dbtz = data[11];

airports.add(new Airport(airportID, name, city, country,

code3, code4, lat, lon, alt, tz, dst, dbtz));

}

br.close();

return airports;

}

// Linear search for the airport code

public static String findAirportCode(String toFind,

ArrayList<Airport> airports)

{

int index = 0;

while (index < airports.size()) {

if (toFind.equals(airports.get(index).getCity())) {

return airports.get(index).getCode3();

}

index++;

}

return null;

}

// Binary search for the airport code

// toFind is a city name

public static String findAirportCodeBS(String toFind, ArrayList<Airport> airports) {

int low = 0;

int high = airports.size()-1;

int mid;

while (low <= high) {

mid = low + ((high-low)/2);

int compare = toFind.compareTo(airports.get(mid).getCity());

if (compare < 0) {

high = mid - 1;

}

else if (compare > 0) {

low = mid+1;

}

else return (airports.get(mid)).getCode3();

}

return null;

}

/\*\* Sort an array of ints using Selection Sort \*/

public static void selectionSort( int[] vals )

{

int minI;

for ( int i=0; i < vals.length-1 ; i++ ) {

minI = i;

for ( int j=i; j < vals.length; j++ ) {

if ( vals[j] < vals[minI] ) {

minI = j ;

}

}

swap(vals, minI, i);

}

}

private static void swap(int[] vals, int ind1, int ind2)

{

int temp = vals[ind1];

vals[ind1] = vals[ind2];

vals[ind2] = temp;

}

/\*\* Sort an array of ints using a mystery algorithm \*/

public static void mysterySort( int[] vals ) {

int currInd;

for ( int pos=1; pos < vals.length ; pos++ ) {

currInd = pos ;

while ( currInd > 0 &&

vals[currInd] < vals[currInd-1] ) {

swap(vals, currInd, currInd-1);

currInd = currInd - 1;

}

}

}

public static void main(String[] args) throws IOException

{

String datafile = "data/airports.dat";

ArrayList<Airport> airports = readFile(datafile);

Collections.sort(airports);

System.out.println(findAirportCodeBS("Å½ilina", airports));

int[] vals = new int[20];

Random r = new Random();

System.out.println("Unsorted ints are...");

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

{

vals[i] = r.nextInt(100); // Randomly generated int less than 100

System.out.print(vals[i] + " ");

}

System.out.println("\nSorted ints are...");

mysterySort(vals);

//selectionSort(vals);

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

{

System.out.print(vals[i] + " ");

}

}

}

**package** demos;

/\*\* A class to represent Airport data.\*/

**public** **class** Airport **implements** Comparable<Airport> {

**private** **int** airportID;

**private** String name;

**private** String city;

**private** String country;

**private** String code3;

**private** String code4;

**private** **double** latitude;

**private** **double** longitude;

**private** **int** altitude;

**private** **float** timezone;

**private** **char** dst;

**private** String dbTimezone;

**public** Airport(**int** airportID, String name, String city, String country, String code3, String code4, **double** latitude,

**double** longitude, **int** altitude, **float** timezone, **char** dst, String dbTimezone) {

**this**.airportID = airportID;

**this**.name = name;

**this**.city = city;

**this**.country = country;

**this**.code3 = code3;

**this**.code4 = code4;

**this**.latitude = latitude;

**this**.longitude = longitude;

**this**.altitude = altitude;

**this**.timezone = timezone;

**this**.dst = dst;

**this**.dbTimezone = dbTimezone;

}

**public** **int** getAirportID() {

**return** airportID;

}

**public** String getName() {

**return** name;

}

**public** String getCity() {

**return** city;

}

**public** String getCountry() {

**return** country;

}

**public** String getCode3() {

**return** code3;

}

**public** String getCode4() {

**return** code4;

}

**public** **double** getLatitude() {

**return** latitude;

}

**public** **double** getLongitude() {

**return** longitude;

}

**public** **int** getAltitude() {

**return** altitude;

}

**public** **float** getTimezone() {

**return** timezone;

}

**public** **char** getDst() {

**return** dst;

}

**public** String getDbTimezone() {

**return** dbTimezone;

}

**public** **int** compareTo(Airport other)

{

**return** **this**.city.compareTo(other.city);

}

}