**Source code**

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package project1;

import java.io.BufferedReader;

import java.io.Console;

import java.io.File;

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.text.DecimalFormat;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.Collection;

import java.util.Date;

import java.util.Iterator;

import java.util.LinkedList;

import java.util.List;

import java.util.ListIterator;

import java.util.Scanner;

import static javafx.application.Platform.exit;

/\*\*

\*

\* @author lucasmarques

\*/

public class Project1 {

/\*\*

\* @param args the command line arguments

\*/

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

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

String input = br.readLine();

LinkedList<Patient> patients = new LinkedList();

File file = new File(input);

try {

Scanner sc = new Scanner(file);

while (sc.hasNext()) {

String line = sc.nextLine();

String[] words = line.split(" ");

int addlenght = words.length; //how much words one line has

int addressEnd = addlenght - 5; //which words is the last word of the address

String address = words[3];

for (int i = 4; i < ((addlenght - 5)); i++) { //split the address from the rest of the line

address = address + " " + words[i];

}

SimpleDateFormat formatter = new SimpleDateFormat("MM/dd/yyyy");

try {

Date birth = formatter.parse(words[addressEnd + 2]); //Creating date variables

Date first = formatter.parse(words[addressEnd + 3]);

Date last = formatter.parse(words[addressEnd + 4]);

Patient patient;

patient = new Patient(words[0] + " " + words[1], words[2], address, Integer.parseInt(words[addressEnd]), Double.parseDouble(words[addressEnd + 1]), birth, first, last);

patients.add(patient);

} catch (Exception e) {

System.out.println(e);

}

}

} catch (FileNotFoundException ex) {

System.out.println("File doesn't exist");

} catch (NumberFormatException e) {

System.out.println("Error");

} catch (Exception e) {

System.out.println("Error");

}

System.out.println("1. Display");

System.out.println("2. Add a new patient");

System.out.println("3. Show information for a patient.");

System.out.println("4. Delete a patient.");

System.out.println("5. Show avarage patient age");

System.out.println("6. Show information for the youngest patient.");

System.out.println("7. Show notification list.");

System.out.println("8. Quit.");

String menu = br.readLine();

Menu m = new Menu();

while (true) {

switch (menu) {

case "1":

m.showsPatients(patients);

break;

case "2":

System.out.println("Enter patient id: "); //looking if id already exist

String id = br.readLine();

if (!m.exists(patients, Integer.parseInt(id))) {

m.newPatient(patients, id);

} else {

System.out.println("Patient already exists");

}

break;

case "3":

m.showsPatients(patients, Integer.parseInt(br.readLine()));

break;

case "4":

int removeId = m.deletePatient(patients, Integer.parseInt(br.readLine()));

if (removeId != -1) { //test if the patient was removed

patients.remove(removeId);

System.out.println("Patient removed.");

} else {

System.out.println("Patient not found.");

}

break;

case "5":

DecimalFormat df = new DecimalFormat("#.#"); //formating value

System.out.println("Average age: " + df.format(m.averageAge(patients)));

break;

case "6":

m.youngestInfo(patients);

break;

case "7":

m.overduePatient(patients);

break;

case "8":

m.quit(patients);

System.exit(0);

break;

}

System.out.println("1. Display");

System.out.println("2. Add a new patient");

System.out.println("3. Show information for a patient.");

System.out.println("4. Delete a patient.");

System.out.println("5. Show avarage patient age");

System.out.println("6. Show information for the youngest patient.");

System.out.println("7. Show notification list.");

System.out.println("8. Quit.");

menu = br.readLine();

}

// TODO code application logic here

}

}

package project1;

import java.io.BufferedReader;

import java.io.FileNotFoundException;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import java.io.UnsupportedEncodingException;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.Date;

import java.util.Iterator;

import java.util.LinkedList;

/\*\*

\*

\* @author lucasmarques

\*/

public class Menu {

public void showsPatients(LinkedList ll) {

Iterator i = sortedList(ll).iterator();

while (i.hasNext()) {

Patient p = (Patient) i.next();

System.out.println(p.getName() + ", " + Integer.parseInt(p.getId()));

}

}

public void newPatient(LinkedList<Patient> patients, String id) throws IOException, ParseException {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

SimpleDateFormat formatter = new SimpleDateFormat("MM/dd/yyyy"); //adding a new patient

Patient p = new Patient();

System.out.println("Enter patient name: ");

p.setName(br.readLine());

p.setId(id);

System.out.println("Enter address: ");

p.setAddress(br.readLine());

System.out.println("Enter patient height: ");

p.setHeight(Integer.parseInt(br.readLine()));

System.out.println("Enter patient weight: ");

p.setWeight(Double.parseDouble(br.readLine()));

System.out.println("Enter patient birthday (MM/DD/YYYY): ");

p.setBirth(formatter.parse(br.readLine()));

System.out.println("Enter patient first visit (MM/DD/YYYY): ");

p.setFirstVisit(formatter.parse(br.readLine()));

System.out.println("Enter patient last visit (MM/DD/YYYY): ");

p.setLastVisit(formatter.parse(br.readLine()));

patients.add(p);

}

public void showsPatients(LinkedList<Patient> patients, int id) {

Iterator i = patients.iterator();

Patient p = null;

while (i.hasNext()) { //searching for the patient

p = (Patient) i.next();

if (id == Integer.parseInt(p.getId())) {

break;

} else {

p = null;

}

}

if (p != null) {

System.out.println(p.getName() + ", " + p.getId() + ", " + p.getAddress() + ", " + p.getHeight() + "ft " + p.getWeight() + "in");

if(p.get\_time\_as\_patient() < 1)

System.out.println("Less than a Year.");

else

System.out.println("Years as patient: " + p.get\_time\_as\_patient());

} else {

System.out.println("Patient not found.");

}

}

public LinkedList sortedList(LinkedList<Patient> ll) {

LinkedList patients = (LinkedList) ll.clone(); //organizing the list

LinkedList a = new LinkedList();

while (!patients.isEmpty()) {

Iterator i = patients.iterator();

Patient first = (Patient) i.next();

while (i.hasNext()) {

Patient now = (Patient) i.next();

if (Integer.parseInt(first.getId()) > Integer.parseInt(now.getId())) {

first = now;

}

}

a.add(first);

patients.remove(first);

}

return a;

}

public int deletePatient(LinkedList<Patient> patients, int id) {

Iterator i = patients.iterator();

int toRemove = -1; //flag that represents if the list doesn't contain the patient

Patient p = null;

while (i.hasNext()) {

p = (Patient) i.next();

if (Integer.parseInt(p.getId()) == id) {

toRemove = patients.indexOf(p);

}

}

return toRemove;

}

public float averageAge(LinkedList<Patient> patients) {

if (!patients.isEmpty()) {

Iterator i = patients.iterator();

float average = 0, numPatients = 0;

while (i.hasNext()) {

numPatients++;

Patient p = (Patient) i.next();

average = average + p.get\_age();

}

return (average / numPatients);

}else{

System.out.println("Empty List.");

return 0;

}

}

void youngestInfo(LinkedList<Patient> patients) {

Iterator i = patients.iterator();

int age = 0, youngest = 0, id = 0, youngid = 0;

Patient young, p;

if (i.hasNext()) {

young = p = (Patient) i.next();

age = youngest = p.get\_age();

} else {

System.out.println("Empty list");

return;

}

while (i.hasNext()) {

p = (Patient) i.next();

id++;

int ageCalc = p.get\_age();

if (age > ageCalc) {

young = p;

youngid = id;

youngest = ageCalc;

}

}

showsPatients(patients, Integer.parseInt(young.getId()));

}

void overduePatient(LinkedList<Patient> patients) {

Iterator i = patients.iterator();

int id = 0;

while (i.hasNext()) {

Patient p = (Patient) i.next();

if (p.get\_time\_since\_last\_visit() > 2) {

System.out.println(p.getName() + ", " + p.getId());

}

id++;

}

}

void quit(LinkedList<Patient> patients) throws FileNotFoundException, UnsupportedEncodingException {

Iterator i = patients.iterator();

PrintWriter writer = new PrintWriter("output.txt", "UTF-8");

while (i.hasNext()) {

Patient p = (Patient) i.next();

writer.println(p.getName() + " " + p.getId() + " " + p.getAddress() + " " + p.getHeight() + " " + p.getWeight() + " " + (p.getBirth().getMonth() + 1) + "/" + p.getBirth().getDate() + "/" + (p.getBirth().getYear()+1900) + " " + (p.getFirstVisit().getMonth()+ 1) + "/" + p.getFirstVisit().getDate() + "/" + (p.getFirstVisit().getYear()+1900) + " " + (p.getLastVisit().getMonth()+ 1) + "/" + p.getLastVisit().getDate() + "/" + (p.getLastVisit().getYear()+1900));

}

writer.close();

}

boolean exists(LinkedList<Patient> Patients, int id) {

Iterator i = Patients.iterator();

while (i.hasNext()) {

Patient p = (Patient) i.next();

if (id == Integer.parseInt(p.getId())) {

return true;

}

}

return false;

}

}

package project1;

import java.util.Date;

/\*\*

\*

\* @author lucasmarques

\*/

public class Patient {

private String name;

private String id;

private String address;

private int height;

private double weight;

private Date birth;

private Date firstVisit;

private Date lastVisit;

Patient(String name, String id, String address, int height, double weight, Date birth, Date firstVisit, Date lastVisit) {

this.name = name;

this.id = id;

this.address = address;

this.height = height;

this.weight = weight;

this.birth = birth;

this.firstVisit = firstVisit;

this.lastVisit = lastVisit;

}

Patient() {

}

public int get\_age() {

Date now = new Date();

int nowMonth = now.getMonth();

int nowYear = now.getYear();

int result = nowYear - birth.getYear();

if (birth.getMonth() > nowMonth) {

result--;

} else if (birth.getMonth() == nowMonth) {

int nowDay = now.getDate();

if (birth.getDate() > nowDay) {

result--;

}

}

return result;

}

public int get\_time\_as\_patient() {

Date now = new Date();

int nowMonth = now.getMonth();

int nowYear = now.getYear();

int result = nowYear - firstVisit.getYear();

if (firstVisit.getMonth() > nowMonth) {

result--;

} else if (firstVisit.getMonth() == nowMonth) {

int nowDay = now.getDate();

if (firstVisit.getDay() > nowDay) {

result--;

}

}

return result;

}

public int get\_time\_since\_last\_visit() {

Date now = new Date();

int nowMonth = now.getMonth();

int nowYear = now.getYear();

int result = nowYear - lastVisit.getYear();

if (lastVisit.getMonth() > nowMonth) {

result--;

} else if (lastVisit.getMonth() == nowMonth) {

int nowDay = now.getDate();

if (lastVisit.getDay() > nowDay) {

result--;

}

}

return result;

}

/\*\*

\* @return the name

\*/

public String getName() {

return name;

}

/\*\*

\* @param name the name to set

\*/

public void setName(String name) {

this.name = name;

}

/\*\*

\* @return the id

\*/

public String getId() {

return id;

}

/\*\*

\* @param id the id to set

\*/

public void setId(String id) {

this.id = id;

}

/\*\*

\* @return the address

\*/

public String getAddress() {

return address;

}

/\*\*

\* @param address the address to set

\*/

public void setAddress(String address) {

this.address = address;

}

/\*\*

\* @return the height

\*/

public int getHeight() {

return height;

}

/\*\*

\* @param height the height to set

\*/

public void setHeight(int height) {

this.height = height;

}

/\*\*

\* @return the weight

\*/

public double getWeight() {

return weight;

}

/\*\*

\* @param weight the weight to set

\*/

public void setWeight(double weight) {

this.weight = weight;

}

/\*\*

\* @return the birth

\*/

public Date getBirth() {

return birth;

}

/\*\*

\* @param birth the birth to set

\*/

public void setBirth(Date birth) {

this.birth = birth;

}

/\*\*

\* @return the firstVisit

\*/

public Date getFirstVisit() {

return firstVisit;

}

/\*\*

\* @param firstVisit the firstVisit to set

\*/

public void setFirstVisit(Date firstVisit) {

this.firstVisit = firstVisit;

}

/\*\*

\* @return the lastVisit

\*/

public Date getLastVisit() {

return lastVisit;

}

/\*\*

\* @param lastVisit the lastVisit to set

\*/

public void setLastVisit(Date lastVisit) {

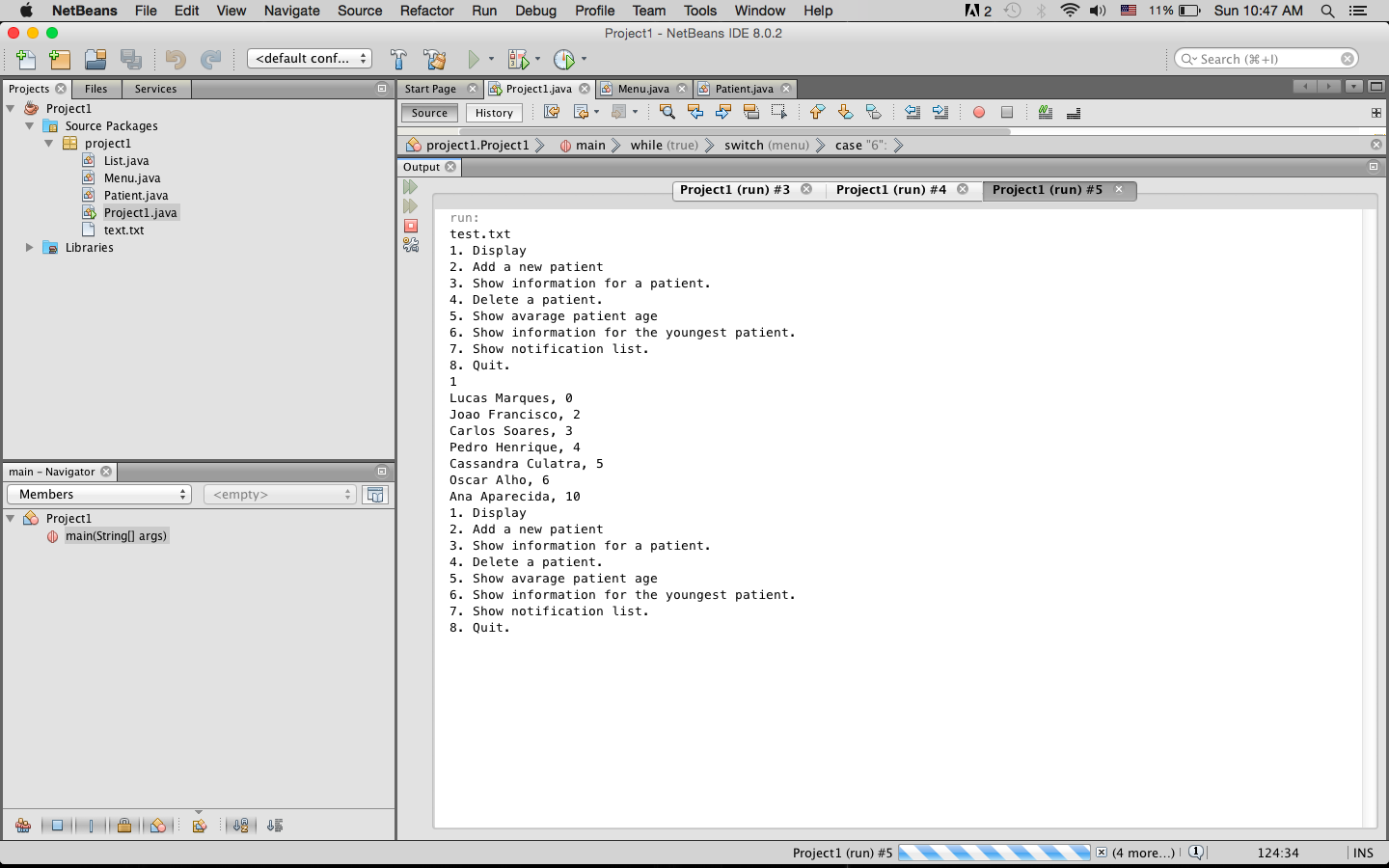
this.lastVisit = lastVisit;

}

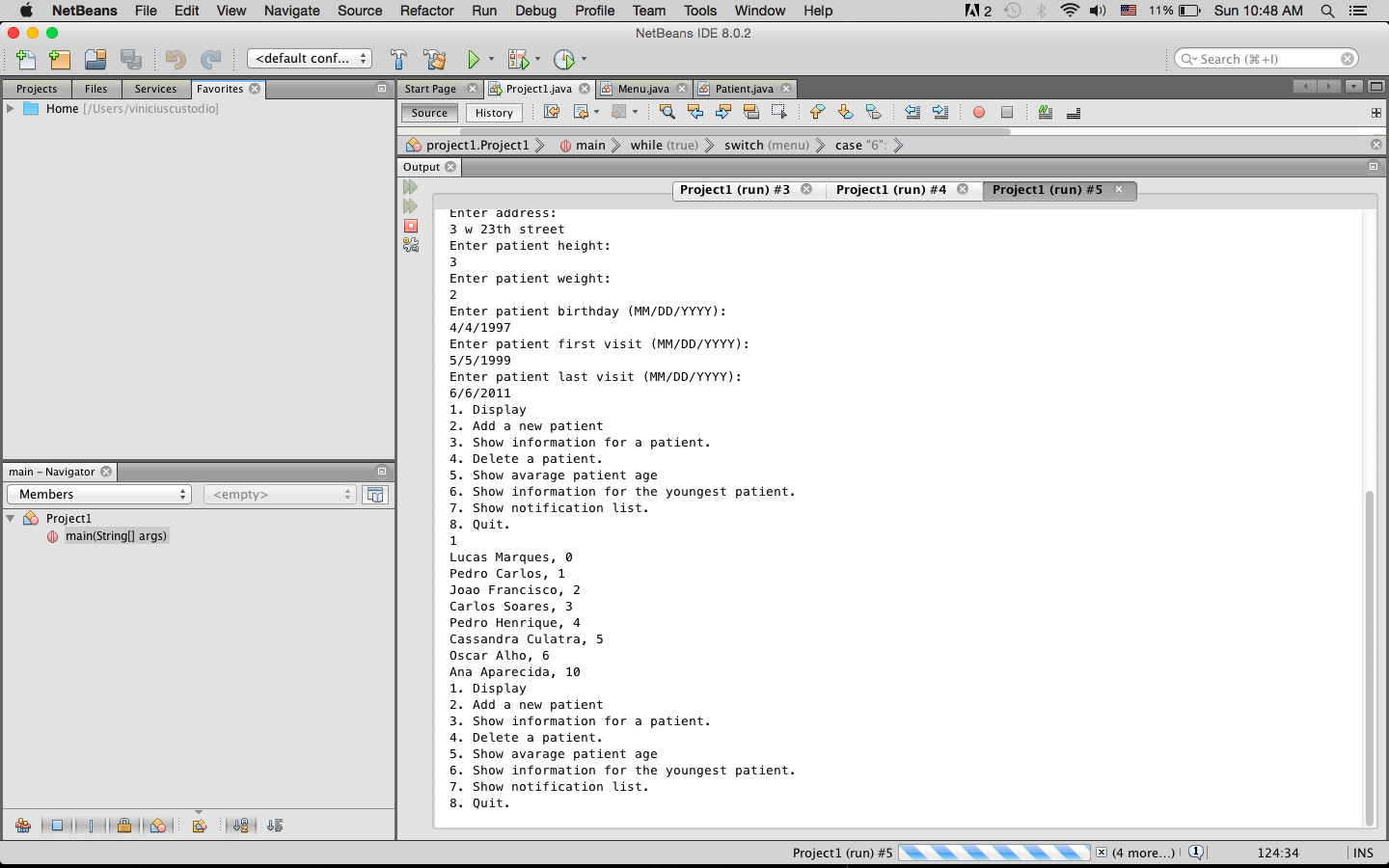
}

**Snapshots**

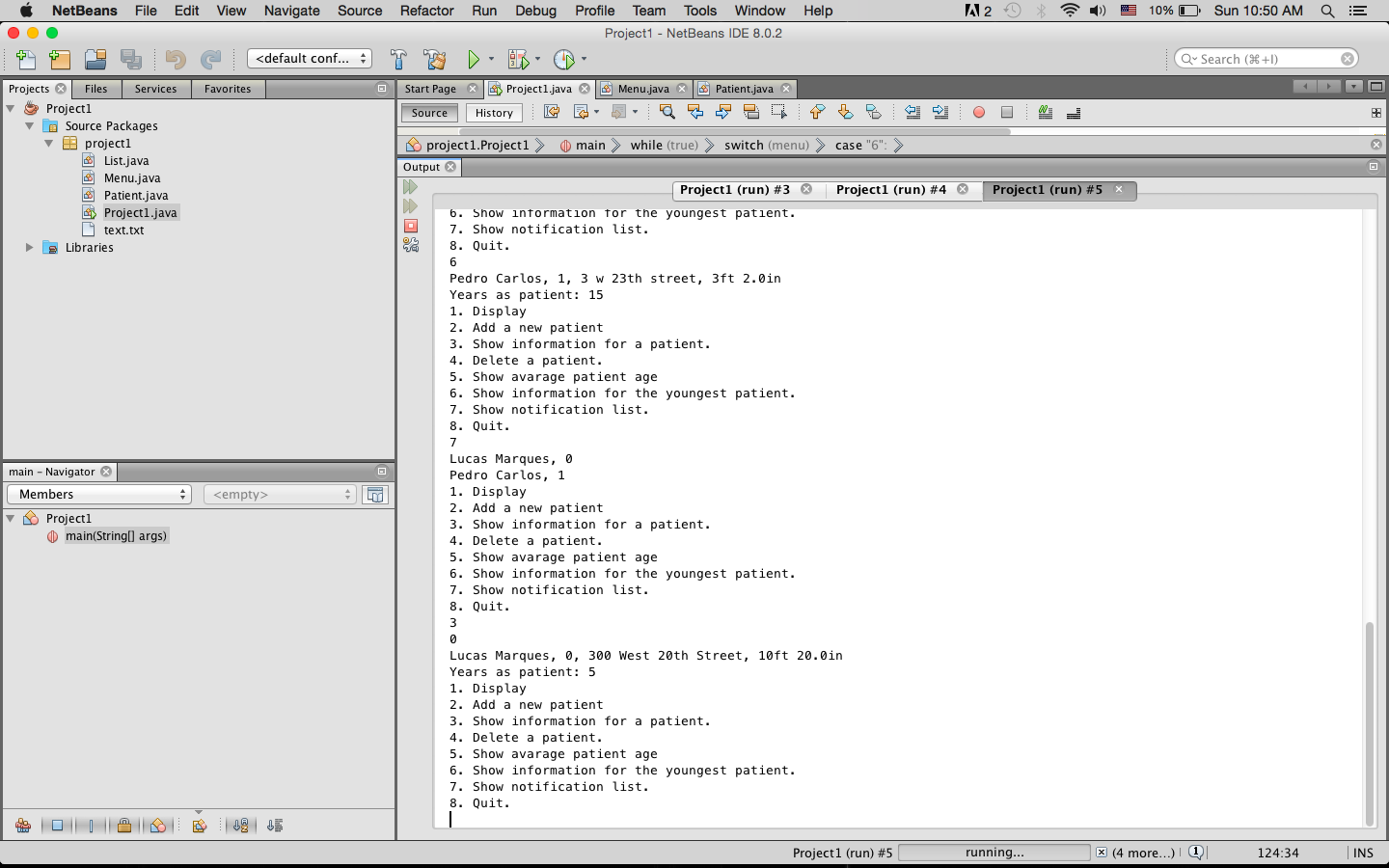
Displaying List of Patients



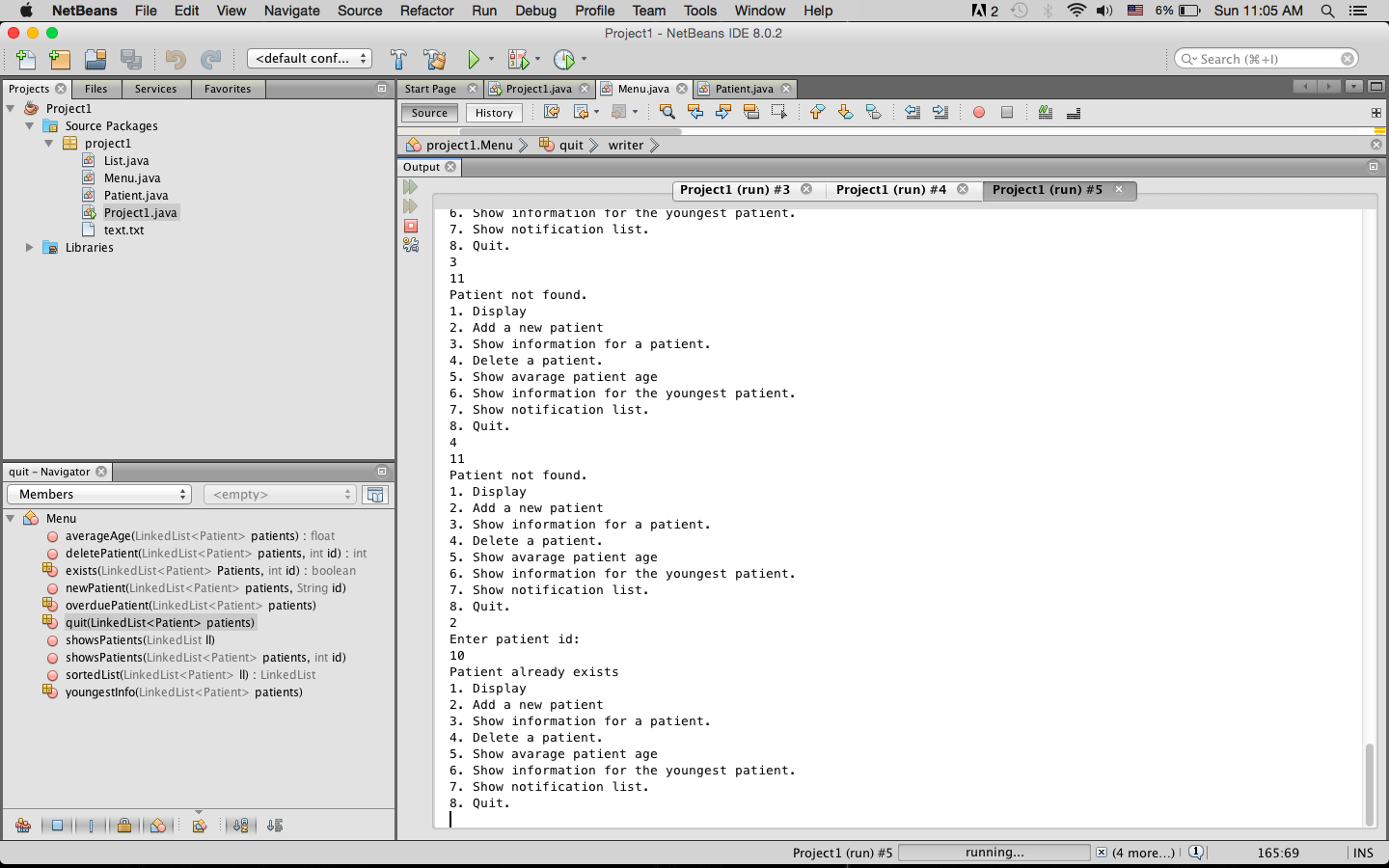
Adding a new Patient



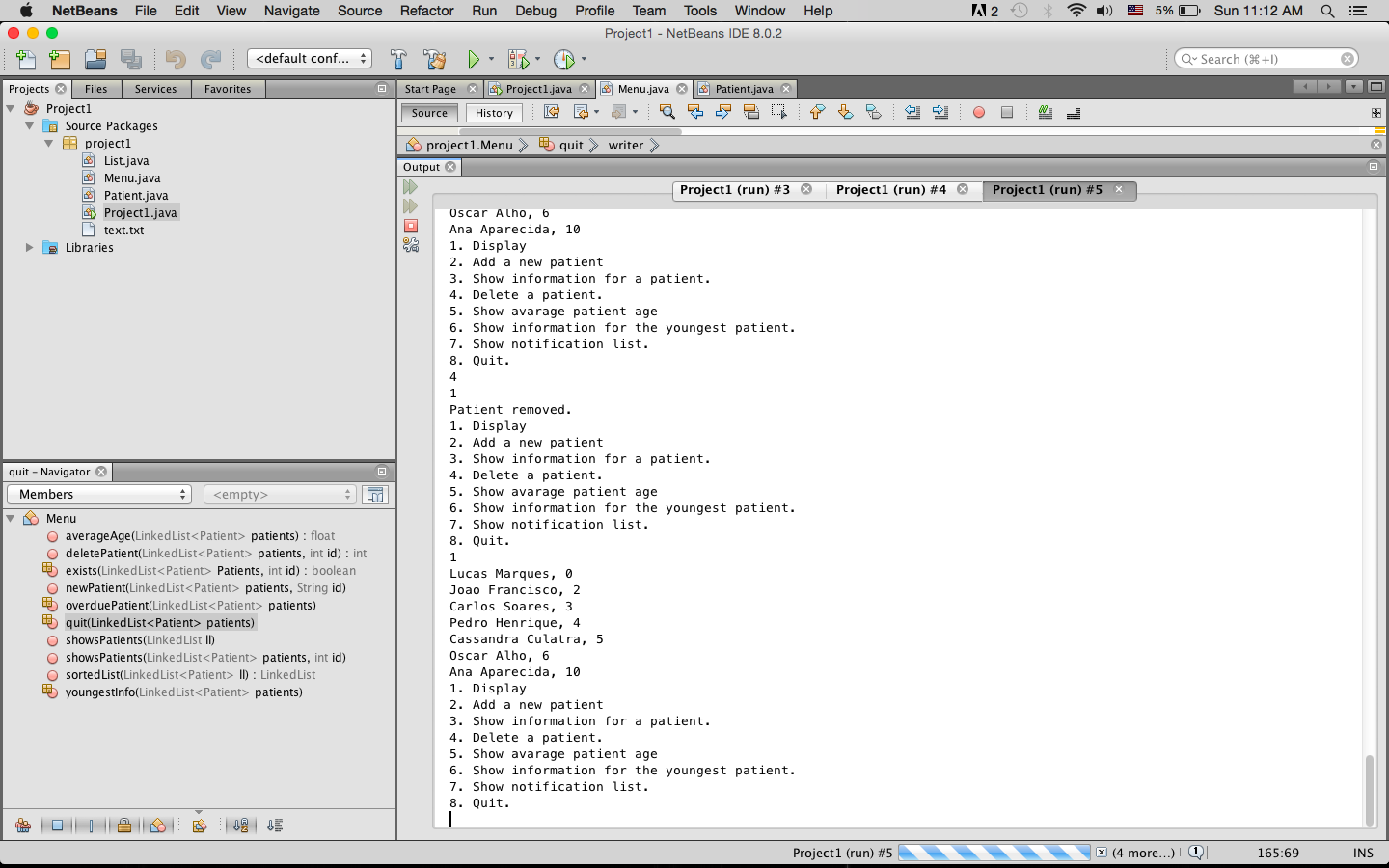
Showing information of the youngest patient, notification list, and information about one patient.

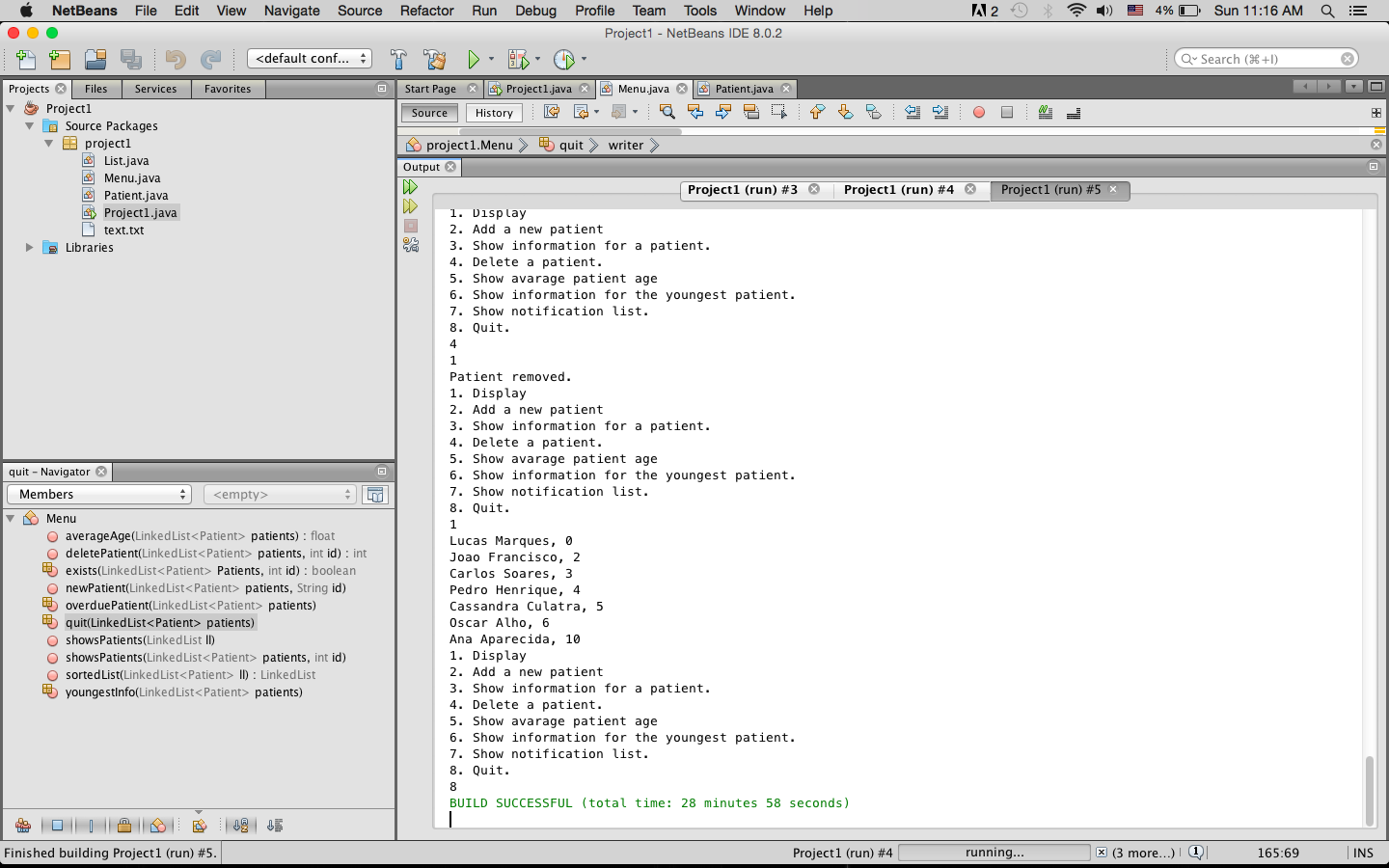


Showing with you look for a patient that doesn’t exist, if you try to delete a patient that also doesn’t exists and if you try to add a new patient with an id that already exist



Deleting a patient



Closing program

In the source code that I submitted at Sunday was an error that I found after the submission time. Instead of use getDate() to get the day of the month I was using getDay() which returns the day of the week.