****



**TRABAJO PRÁCTICO**

**PROGRAMACIÓN II**

segundo cuatrimestre

2020



Licenciatura en Sistemas

Alumnos:

Eros Montevega

dni: 40306303

Franzetti Mariano Andres

dni: 40290786

**Introducción**

El objetivo del trabajo es implementar un programa en lenguaje Java, para ello creamos los procedimientos necesarios para el correcto funcionamiento de la misma. En el desarrollo del informe se explicarán los procedimientos creados, sus funciones y la manera en que fueron implementados dentro del programa para lograr el objetivo. Asimismo se comentarán los problemas ocurridos y sus soluciones.

La finalidad del programa es crear un sistema que pueda ser utilizado en un centro médico por los empleados de la misma y así poder tener una buena organización.

Podríamos decir que un centro médico es aquel donde las personas al momento de atenderse se consideran pacientes y tienen diferentes categorías, en este caso podemos ver que tenemos 3, también vamos a ver que tenemos varios tipos de atenciones que le corresponde a cada tipo de paciente, además cada paciente va a diferenciarse por un número de historia clínica, como también el médico tiene un numero de matricula

En este caso podemos ver cómo la tecnología puede lograr infinidades de cosas, logrando así, otorgar a las personas programas que son capaces de resolver sus problemas en tiempo y forma, de una manera eficiente y práctica, podemos decir entonces, que la tecnología le facilita la vida al ser humano realizando muchas de las tareas, que tiempo atrás eran impensadas. Lo que queremos lograr es que la persona que utilice este programa no tenga que tener una hoja y un lápiz para tener que resolver una operación simple.

## **Esquema general**

CLASES

CLASE CENTRO MÉDICO:

ATRIBUTOS

La clase Centro Médico es la más importante de todas ya que en ella se realizan las mayoría de las funciones más importantes, como agregar médicos, especialidades, atenciones, etc.

En ella contamos con los siguientes atributos:

private HashMap<Integer, Medico> médicos; tiene una colección de médicos en las cuales se puede acceder a uno de ellos en O(1)

private HashMap<String, Double> especialidades; tiene una colección de especialidades de las cuales en el String tiene el nombre y en el Double tiene el precio

private HashMap<Integer, Paciente> pacientes; tiene una colección de pacientes en la cual podemos acceder a uno de ellos en O(1)

private HashMap<String, Double> obrasSociales; tiene una colección de obras sociales en la cual la clave es un String que es el nombre y el Double el descuento

private HashMap<Integer, Boolean> habitaciones; es una colección de habitaciones donde se puede acceder a cada una de ellas en O(1) la variable booleana verificar si la habitación está ocupada o no

private String cuit; tenemos una variable privada de tipo string que tiene el cuit del centro

private String nombre; tenemos una variable privada de tipo string que tiene el nombre del centro médico

private Double costoDiaInternacion; tenemos una variable double con el costo del dia de la internación

METODOS:

public CentroMedico(String nombre, String cuit, Double costoDiaInternacion) throws Exception {

this.cuit = cuit;

this.nombre = nombre;

this.costoDiaInternacion = costoDiaInternacion;

medicos = new HashMap<Integer, Medico>();

pacientes = new HashMap<Integer, Paciente>();

especialidades = new HashMap<String, Double>();

obrasSociales = new HashMap<String, Double>();

habitaciones = new HashMap<Integer, Boolean>();

crearHabitaciones(100);

}

el constructor de centro médico va a recibir por parámetro un nombre, un cuit y un costo de dia de internacion. Dentro del constructor va a tener colecciones de médicos, pacientes, especialidades, obras sociales y va a crear habitaciones mediante un valor fijo entero.

public boolean agregarEspecialidad(String nombre, Double valor) {

if (!especialidades.containsKey(nombre)) {

especialidades.put(nombre, valor);

return true;

}

return false;

}

el metodo agregarEspecialidad recibe por parámetro un nombre y un valor para la especialidad y si no la contiene en la colección la agrega devolviendo true si se agrega correctamente y false si no.

public boolean agregarMedico(String nombre, Integer matricula, String nomEspecialidad, Double honorarios) {

if (!medicos.containsKey(matricula)) {

Medico medico = new Medico(nombre, matricula, nomEspecialidad, honorarios);

medicos.put(matricula, medico);

return true;

}

return false;

}

el metodo agregarEspecialidad recibe por parámetro un nombre, una matrícula, un nombre de especialidad y un honorario, donde se fija que si no hay medicos en la coleccion los agrega.

public boolean agregarPacientePrivado(String nombre, Integer historiaClinica, Fecha nac) {

if (!pacientes.containsKey(historiaClinica)) {

PacientePrivado paciente = new PacientePrivado(nombre, historiaClinica, nac);

pacientes.put(historiaClinica, paciente);

return true;

}

return false;

}

el metodo publico agregarPacientePrivado recibe por parametro un nombre, una historia clinica y una fecha de nacimiento, si el paciente no se encuentra en la coleccion de pacientes crea el paciente privado y lo agrega a la lista de pacientes.

public boolean agregarPacienteAmbulatorio(String nombre, Integer nroHistoriaClinica, Fecha nac) {

if (!pacientes.containsKey(nroHistoriaClinica)) {

PacienteAmbulatorio paciente = new PacienteAmbulatorio(nombre, nroHistoriaClinica, nac);

pacientes.put(nroHistoriaClinica, paciente);

return true;

}

return false;

}

el metodo publico agregarPacienteAmbulatorio recibe un nombre por parametro, un numero de historia clinica y una fecha de nacimiento, en el metodo nos fijamos si existe el paciente en la coleccion de pacientes y si no existe lo crea y lo agrega, si existe retorna false

public boolean agregarPacienteObraSocial(String nombre, Integer historiaClinica, Fecha nac, String obraSocial,

Double descuento) {

if (!pacientes.containsKey(historiaClinica)) {

// si existe se actualiza si no, lo agrega

obrasSociales.put(obraSocial, descuento);

PacienteObraSocial paciente = new PacienteObraSocial(nombre, historiaClinica, nac, obraSocial, descuento);

pacientes.put(historiaClinica, paciente);

return true;

}

return false;

}

en este método público de tipo boolean agregarPacienteAmbulatorio recibimos por parámetro un nombre, número de historia clínica, una fecha de nacimiento, un string con el nombre de obra social y un double con el descuento, se fija si el paciente existe o no en la coleccion y si no existe en la coleccion, crea al paciente de obra social y lo agrega retornando true y false si ya existia y no lo agrega.

public boolean agregarAtencion(Integer historiaClinica, Fecha fecha, Integer matricula) {

if (pacientes.containsKey(historiaClinica)) {

if (validarClasePaciente(pacientes.get(historiaClinica), "PacientePrivado")) {

PacientePrivado paciente = (PacientePrivado) pacientes.get(historiaClinica);

if (medicos.containsKey(matricula)) {

Medico medico = medicos.get(matricula);

if (especialidades.containsKey(medico.obtenerEspecialidad())) {

Double importe = especialidades.get(medico.obtenerEspecialidad());

return paciente.agregarAtencionConsultorio(fecha, medico, importe);

}

}

}

}

return false;

}

agregarAtencion es un metodo publico que recibe por parametro una historia clinica, una fecha y una matricula: en el metodo se fija si el paciente existe el paciente en la colección y tambien valida el tipo de paciente y si es de metodo privado, se instancia al paciente de tipo privado, luego nos fijamos si el medico existe y si la especialidad tambien existe, de darse todas estas condicienes se agrega la atencion con una fecha, un medico y un importe, sino retorna false

// atencion guardia

public boolean agregarAtencion(Integer historiaClinica, Fecha fecha) {

if (pacientes.containsKey(historiaClinica)) {

if (validarClasePaciente(pacientes.get(historiaClinica), "PacientePrivado")) {

PacientePrivado paciente = (PacientePrivado) pacientes.get(historiaClinica);

return paciente.agregarAtencionGuardia(fecha);

}

}

return false;

}

el metodo publico de tipo boolean agregarAtencion es un metodo que agrega las atenciones por guardia que recibe por parametro un numero de historia clinica y una fecha, se fija si el paciente existe en la coleccion de pacientes, luego tambien se verifica el tipo de paciente, y si corresponde el tipo de paciente se instancia al paciente y se agrega la atencion del paciente con la fecha.

private void crearHabitaciones(Integer cantidad) throws Exception {

try {

if (habitaciones.isEmpty()) {

for (int i = 1; i < cantidad; i++) {

habitaciones.put(i, false);

i++;

}

}

} catch (Exception e) {

throw new Exception("fallo al crear habitaciones");

}

}

es un metodo privado de tipo void crearHabitaciones que recibe por parametro un numero de habitaciones y se fija que si no hay habitaciones a traves de un for y el numero recibido por parametro, las cree.

private Integer obtenerHabitacionVacia() throws Exception {

if (habitaciones.containsValue(false) && !habitaciones.isEmpty()) {

Integer cont = 0;

Iterator<Boolean> iterator = habitaciones.values().iterator();

while (iterator.hasNext()) {

cont++;

if ((Boolean) iterator.next() == false) {

break;

}

;

}

return cont;

} else {

throw new Exception("No hay habitaciones vacias o esta vacio");

}

}

obtenerHabitacionVacia() es un metodo de tipo privado que se fija si por lo menos hay una habitacion vacia y tambien si hay habitaciones, luego itera y te devuelve la primera habitacion vacia que encuentra.

public boolean agregarInternacion(Integer historiaClinica, String area, Fecha fechaIngreso) throws Exception {

if (pacientes.containsKey(historiaClinica)) {

if (validarClasePaciente(pacientes.get(historiaClinica), "PacienteObraSocial")) {

PacienteObraSocial paciente = (PacienteObraSocial) pacientes.get(historiaClinica);

return paciente.agregarInternacion(area, fechaIngreso, costoDiaInternacion, obtenerHabitacionVacia());

}

}

return false;

}

el metodo publico boolean agregarInternacion recibe por parametro un numero de historiaclinica, un nombre de area, una fecha de ingreso, donde primero se fija si exite el paciente en la coleccion de pacientes y luego valida si es de clase paciente obra social, luego instancia al paciente y lo agrega si no cumple con las condiciones retorna false

public boolean altaInternacion(Integer historiaClinica, Fecha fechaAlta) throws ParseException {

if (pacientes.containsKey(historiaClinica)) {

if (validarClasePaciente(pacientes.get(historiaClinica), "PacienteObraSocial")) {

PacienteObraSocial paciente = (PacienteObraSocial) pacientes.get(historiaClinica);

return paciente.darAlta(fechaAlta);

}

}

return false;

}

es un metodo publico de tipo booleano altaInternacion que recibe por parametro un numero de historia clinica, una fecha de alta, en el metodo se fija si el paciente existe en la coleccion de paciente luego valida si es de clase paciente obrasocial y si se cumple con el requisito se instancia al paciente de obra social y se lo da de alta, si no cumple con las condiciones retorna false

public boolean agregarTratamiento(Integer historiaClinica, Integer matricula, String tratamiento) {

if (pacientes.containsKey(historiaClinica)) {

if (validarClasePaciente(pacientes.get(historiaClinica), "PacienteAmbulatorio")) {

PacienteAmbulatorio paciente = (PacienteAmbulatorio) pacientes.get(historiaClinica);

return paciente.agregarTratamiento(medicos.get(matricula), tratamiento);

}

}

return false;

}

el metodo publico de tipo booleano agregarTratamiento recibe un numero de histria clinica por parametro, un numero de matricula y un nombre de tratamiento, en el metodo se fija si el paciente existe en la coleccion de paciente, luego valida de que sea de tipo ambulatorio y luego lo instancia, al final lo agrega y retorna true, si no se cumple la condicion retorna falso

public Double getSaldo(Integer historiaClinica) throws Exception {

if (pacientes.containsKey(historiaClinica)) {

Paciente paciente = pacientes.get(historiaClinica);

return paciente.obtenerSaldo();

}

throw new Exception("No existe el paciente");

}

el metodo publico getSaldo recibe por parametro un numero de historia clinica, en el metodo nos fijamos si existe el paciente en la coleecion de paciente, luego se lo instancia y retorna el saldo del paciente, si no cumple con la condicion retorna una excepcion.

public void pagarSaldo(Integer historiaClinica) throws Exception {

if (pacientes.containsKey(historiaClinica)) {

Paciente paciente = pacientes.get(historiaClinica);

paciente.pagarSaldo();

} else {

throw new Exception("No existe el paciente");

}

}

el metodo publico void pagarSaldo recibe por parametro un numero de historia clinica y en el metodo se fija si existe el paciente en la coleccion de pacientes, si existe luego se lo instancia y se utilza la funcion pagar saldo, si no tira una excepcion

public List<Integer> listaInternacion() throws Exception {

if (pacientes.isEmpty()) {

throw new Exception("no hay pacientes");

}

List<Integer> list = new ArrayList<Integer>();

for (Paciente paciente : pacientes.values()) {

if (validarClasePaciente(paciente, "PacienteObraSocial")) {

PacienteObraSocial pacienteOS = (PacienteObraSocial) paciente;

if (pacienteOS.hayInternacionActiva()) {

list.add(paciente.nroHistoriaClinica);

}

}

}

return list;

}

es un metodo publico que devuelte una lista de listadeinternaciones, se fija si pacientes esta vacion si no, crea una lista y se recorre una lista de pacientes y valida que el paciente sea de obrasocial, luego se lo instancia, y al final se fija si la internacion esta activa y si esta activa lo agrega a la lista.

public Map<Fecha, String> atencionesEnConsultorio(Integer historiaClinica) {

Map<Fecha, String> atencionesConsultorio = new HashMap<Fecha, String>();

if (pacientes.containsKey(historiaClinica)) {

if (validarClasePaciente(pacientes.get(historiaClinica), "PacientePrivado")) {

PacientePrivado paciente = (PacientePrivado) pacientes.get(historiaClinica);

atencionesConsultorio = paciente.obtenerAtencionesConsultorio();

}

}

return atencionesConsultorio;

}

el metodo publico atencionesenconsultorio devuelve un map con una Fecha y un string, que recibe por parametro un numero de historiaclinica, adentro del metodo se cea el map de atencions en consultorio, luego se fija si existe el paciente en la coleccion de pacientes, tambien verifica si la clase corresponde a pacientePrivado, luego se instancia al paciente, y luego se instancia a las atenciones en consultorio y se obtiene un pacientedeconsultorio, al final retorna las atenciones de consultorio.

private boolean validarClasePaciente(Paciente paciente, String tipo) {

if (paciente.getClass().getSimpleName().equals(tipo)) {

return true;

} else {

return false;

}

}

es un metodo privado de tipo booleano que valida la clase de paciente y recibe por parametro un paciente y un tipo. Se valida que la clase del paciente corresponda con el string tipo y si son iguales recibe true y si no lo son retorna false

CLASE MEDICO

ATRIBUTOS

protected Double importe;

protected Fecha fecha;

protected Boolean pagado;

tiene una variable protegida de tipo doble con el importe

tiene una variable protegida de tipo fecha con una fecha

tiene una variable protegida de tipo booleano pagado que se inicializa en false

METODOS:

public Atencion(Double importe, Fecha fecha) {

this.importe = importe;

this.fecha = fecha;

pagado = false;

}

el construcion de atencion recibe un importe y una fecha de atencion

luego adentro del constructor recibe la variable pagado en false

public Double obtenerImporte() {

return importe;

}

se obtiene el importe de la atencion

public Fecha obtenerFecha() {

return fecha;

}

se obtiene la fecha de la atencion

public Boolean pagar() {

return pagado = true;

}

paga la atencion y retorna true

CLASE CONSULTORIO HEREDADA DE ATENCION

private String nombreMedico; tiene una variable privada de tipo string con un nombre de medico

private String especialidad; tiene una variable privada con un nombre de especialidad

public Consultorio(Double importe, Fecha fecha, String nombreMedico, String especialidad) {

super(importe, fecha);

this.nombreMedico = nombreMedico;

this.especialidad = especialidad;

}

tiene un constructor con un importe, una fecha, un nombre de médico y un nombre de especialidad:

recibe un super que lo hereda de atención y le pasa el importe y la fecha, luego también tiene un nombre de médico y también una especialidad.

public String obtenerEspecialidad() {

return especialidad;

}

en el método obtener Especialidad retorna una especialidad.

CLASE FECHA

public class Fecha {

Integer dia;

Integer mes;

Integer anio;

tiene una variable integer de dia

tiene una variable integer de mes

tiene una variable integer de un año

public Fecha(Integer dia, Integer mes, Integer anio) {

this.dia = dia;

this.mes = mes;

this.anio = anio;

}

tiene un constructor con un parametro de dia, mes y año

public boolean equals(Object obj) {

if (this == obj)

return true;

if (!(obj instanceof Fecha))

return false;

Fecha other = (Fecha) obj;

if (anio == null) {

if (other.anio != null)

return false;

} else if (!anio.equals(other.anio))

return false;

if (dia == null) {

if (other.dia != null)

return false;

} else if (!dia.equals(other.dia))

return false;

if (mes == null) {

if (other.mes != null)

return false;

} else if (!mes.equals(other.mes))

return false;

return true;

}

implementamos un equals para comparar dos objetos de tipo fecha

public String obtenerFecha() {

StringBuilder str = new StringBuilder();

str.append(dia);

str.append("/");

str.append(mes);

str.append("/");

str.append(anio);

return str.toString();

}

este metodo te retorna la fecha

public static Fecha hoy() {

LocalDate fechaAux = LocalDate.now();

return new Fecha(fechaAux.getDayOfMonth(), fechaAux.getMonthValue(), fechaAux.getYear());

}

este metodo retorna la fecha de hoy

public boolean esMayorIgual(Fecha fecha) throws ParseException {

SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");

StringBuilder fechaInicialStr = new StringBuilder();

fechaInicialStr.append(this.anio);

fechaInicialStr.append("-");

fechaInicialStr.append(this.mes);

fechaInicialStr.append("-");

fechaInicialStr.append(this.dia);

StringBuilder fechaFinalStr = new StringBuilder();

fechaFinalStr.append(fecha.anio);

fechaFinalStr.append("-");

fechaFinalStr.append(fecha.mes);

fechaFinalStr.append("-");

fechaFinalStr.append(fecha.dia);

Date fechaInicial = new Date();

Date fechaFinal = new Date();

fechaInicial = dateFormat.parse(fechaInicialStr.toString());

fechaFinal = dateFormat.parse(fechaFinalStr.toString());

int dias = (int) ((fechaFinal.getTime() - fechaInicial.getTime()) / 86400000);

if (dias >= 0) {

return true;

} else {

return false;

}

}

este metodo recibe por parámetro una fecha y devuelve true si la fecha actual es mayor a la anterior y false si no lo es

public int obtenerDias(Fecha fecha) throws ParseException {

SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");

StringBuilder fechaInicialStr = new StringBuilder();

fechaInicialStr.append(this.anio);

fechaInicialStr.append("-");

fechaInicialStr.append(this.mes);

fechaInicialStr.append("-");

fechaInicialStr.append(this.dia);

StringBuilder fechaFinalStr = new StringBuilder();

fechaFinalStr.append(fecha.anio);

fechaFinalStr.append("-");

fechaFinalStr.append(fecha.mes);

fechaFinalStr.append("-");

fechaFinalStr.append(fecha.dia);

Date fechaInicial = new Date();

Date fechaFinal = new Date();

fechaInicial = dateFormat.parse(fechaInicialStr.toString());

fechaFinal = dateFormat.parse(fechaFinalStr.toString());

int dias = (int) ((fechaFinal.getTime() - fechaInicial.getTime()) / 86400000);

return dias;

}

el metodo obtenerDias hace la cuenta de una fecha final menos la fecha inicial y te devuelve la cantidad de dias.

CLASE INTERNACION HEREDADA DE ATENCION

private String area;

private Double descuento;

private Integer nroHabitacion;

private Fecha fechaAlta;

tiene una variable privado de stria para el area de internacion, una variable privada de tipo double con el descuente, una privada de tipo integer con el numero de habitacion y una privada con la fecha de la fecha de alta

METODOS

public Internacion(Double importe, Fecha fecha, String obraSocial, String area, Double descuento,

Integer nroHabitacion) {

super(importe, fecha);

this.obraSocial = obraSocial;

this.area = area;

this.descuento = descuento;

this.nroHabitacion = nroHabitacion;

this.fechaAlta = null;

}

tiene un constructor que recibe por parámetro un importe, una fecha, una obra social, un area, un descuento y un numero de habitacion, tiene un super que hereda de atencion y le pasa un importe y una fecha

public Fecha obtenerFechaAlta() {

return fechaAlta;

}

este metodo devuelve la fecha de alta

public Fecha obtenerFechaIngreso() {

return fecha;

}

este metodo devuelve la fecha de ingreso

public void agregarFechaAlta(Fecha fechaAlta) {

this.fechaAlta = fechaAlta;

}

este metodo agregarfecha alta, le agrega la fecha de alta

public void agregarImporte(Double importe) {

this.importe = importe;

}

el metodo public void agregarImporte recibe por parámetro un importa y el importe le agrega el importe

public Double obtenerImporte() {

return importe;

}

este metodo obtiene el importe de la internacion

CLASE MEDICO

private Integer nroMatricula;

private String nombre;

private Double honorarios;

private String especialidad;

La clase medico tiene tributos de tipo privados de tipo integer con el numero de matricula, un string con un nombre, un doble con honorarios y un string con la especialidad

public Medico(String nombre, Integer nroMatricula, String especialidad, Double honorarios) {

this.nombre = nombre;

this.nroMatricula = nroMatricula;

this.especialidad = especialidad;

this.honorarios = honorarios;

}

el constructor medico recibe por parametro un nombre, un integer con una matricula, un string de especialdad y un double con honorarios.

public Double obtenerHonorarios() {

return honorarios;

}

este metodo obtenerHonorarios retorno honorarios

public String obtenerEspecialidad() {

return especialidad;

}

este metodo obtener especialidad retorno la especialidad

public Integer obtenerMatricula() {

return nroMatricula;

}

este metodo retorna la matricula del medico

public String obtenerNombre() {

return nombre;

}

este metodo retorna el nombre del medico

public boolean equals(Object obj) {

if (this == obj)

return true;

if (obj == null)

return false;

if (getClass() != obj.getClass())

return false;

Medico other = (Medico) obj;

if (nroMatricula == null) {

if (other.nroMatricula != null)

return false;

} else if (!nroMatricula.equals(other.nroMatricula))

return false;

return true;

}

este metodo compara objetos de tipo medico y devuelve true o false

CLASE PACIENTE

protected int nroHistoriaClinica;

protected Fecha nacimiento;

protected String nombre;

protected Double saldo;

la clase paciente tiene variables protegidas, de tipo int con un numero de historia clinica, una fecha de nacimiento, un string con el nombre y un double con un saldo

@Override

public boolean equals(Object obj) {

if (this == obj)

return true;

if (!(obj instanceof Paciente))

return false;

Paciente other = (Paciente) obj;

if (nacimiento == null) {

if (other.nacimiento != null)

return false;

} else if (!nacimiento.equals(other.nacimiento))

return false;

if (nombre == null) {

if (other.nombre != null)

return false;

} else if (!nombre.equals(other.nombre))

return false;

if (nroHistoriaClinica != other.nroHistoriaClinica)

return false;

return true;

}

implementamos un método equals para comparar objetos de tipo paciente

public Paciente(String nombre, Integer nroHistoriaClinica, Fecha nacimiento) {

this.nroHistoriaClinica = nroHistoriaClinica;

this.nacimiento = nacimiento;

this.nombre = nombre;

this.saldo = 0.0;

}

tiene un constructor que recibe por parametro un nombre, un numero de historia clinica y una fecha de nacimiento y adentro del constructor tiene un saldo que se inicializa en 0.0

public Integer obtenerNroHistoriaClinica() {

return nroHistoriaClinica;

}

este metodo devuelve el numero de historia clinica

public Fecha obtenerFechaNacimiento() {

return nacimiento;

}

este metodo devuelve una fecha de nacimiento

public Integer obtenerEdad() throws ParseException {

return nacimiento.obtenerDias(Fecha.hoy()) / 365;

}

este metodo devuelve la edad del paciente

public double obtenerSaldo() {

return saldo;

}

este metodo obtiene el saldo del paciente

public abstract void pagarSaldo() throws Exception;

este metodo de tipo abstracto lo que hace es obligar a las clases hija a que lo implemente para crear la funcionalidad de pagar saldo

CLASE PACIENTE AMBULATORIO HEREDADA DE PACIENTE

private HashMap<Medico, HashMap<String, Boolean>> tratamientos;

Tiene un hashmap de tipo privado donde recibe un medico y adentro del hashmap en la parte de valor tiene un hashmap con un string de nombre y un booleano

METODOS

public PacienteAmbulatorio(String nombre, Integer nroHistoriaClinica, Fecha nacimiento) {

super(nombre, nroHistoriaClinica, nacimiento);

tratamientos = new HashMap<Medico, HashMap<String, Boolean>>();

}

el constructor pacienteambulatorio recibe por parametro un nombre, un numero de historia clinica y una fecha de nacimiento, luego tiene un super que hereda de paciente donde le pasa un nombre, un numero de historia clinica, un nacimiento y luego tiene un hashmap de tratamientos.

public boolean agregarTratamiento(Medico medico, String tratamiento) {

if (tratamientos.containsKey(medico)) {

HashMap<String, Boolean> tratamientosXMedico = tratamientos.get(medico);

if (!tratamientosXMedico.containsKey(tratamiento)) {

saldo = saldo + medico.obtenerHonorarios();

return tratamientosXMedico.put(tratamiento, false);

}

} else {

HashMap<String, Boolean> tratamientosXMedicoNew = new HashMap<String, Boolean>();

tratamientosXMedicoNew.put(tratamiento, false);

tratamientos.put(medico, tratamientosXMedicoNew);

saldo = saldo + medico.obtenerHonorarios();

return true;

}

return false;

}

El metodo publico de tipo booleano agregarTratamiento recibe por parametro un medico y un nombre de tratamiento, luego se fija si en tratamientos existe el medico en el caso de existir el medico, obtiene un lista de los tratamientos de ese medico y revisa que el nombre del tratamiento no exista, si existe retorna false. Por ultimo si el medico no existe en la coleccion lo agrega y crea la coleccion de tratamientos asociada a ese medico con el nuevo tratamiento. Siempre al agregarlo correctamente se modifica el saldo.

public void pagarSaldo() throws Exception {

if (!tratamientos.isEmpty()) {

for (HashMap<String, Boolean> tratamientosXMedicos : tratamientos.values()) {

for (String tratamiento : tratamientosXMedicos.keySet()) {

tratamientosXMedicos.put(tratamiento, true);

}

}

saldo = 0.0;

}

else {

throw new Exception("No hay atenciones para pagar");

}

}

el metodo publico void pagarSaldo, lo que hace es fijarse si tratamientos no esta vacia, si no lo esta recorre un hashmap de los medicos y luego recorre los tratamientos de ese medico y pone los tratamientos en true lo que hace referencia al pago del mismo, y al final pone el saldo en cero, en caso de que no haya tratamientos devuelve una excepcion

CLASE PACIENTEOBRASOCIAL HEREDADA DE PACIENTE

private String obraSocial;

private Double descuento;

private LinkedList<Internacion> internaciones;

Tiene variables de privadas de: tipo string obraSocial, un double descuento que tiene un descuento, luego una lista privada de de linkedlist de internaciones

public PacienteObraSocial(String nombre, Integer nroHistoriaClinica, Fecha nacimiento, String obraSocial,

Double descuento) {

super(nombre, nroHistoriaClinica, nacimiento);

this.obraSocial = obraSocial;

this.descuento = descuento;

internaciones = new LinkedList<Internacion>();

}

un constructor de pacienteObraSocial que recibe por parametro un nombre, un integer con numero de historia clinica, luego una fecha de nacimiento, un string con obrasocial y luego un double con un descuento, en el constructor adentro tengo un super que heredo de paciente que le pasa por parametro un nombre, un numero de historia clinica y un nacimiento. luego las demas variables y se inicializa un linkedlist de internaciones

public String obtenerObraSocial() {

return this.obraSocial;

}

este metodo te devuelve la obraSocial

private boolean contieneFechaIngreso(Fecha fechaIngreso) {

boolean alguno = false;

for (Internacion internacion : internaciones) {

alguno = alguno || (internacion.fecha.equals(fechaIngreso));

}

return alguno;

}

el metodo privado booleano contienefechaingreso recibe una fecha de ingreso, en el metodo usamos acumuladores booleanos y recorremos las internaciones y devolvemos si true alguna de las internaciones coincide con la internacion pasada por parametro

public boolean hayInternacionActiva() {

if(internaciones.isEmpty()) {

return false;

}

return internaciones.getLast().obtenerFechaAlta() == null;

}

el metodo publico booleano hayinternacion activa devuelve true si hay internacion activa y false si no lo esta.

public boolean agregarInternacion(String area, Fecha fechaIngreso, Double importe, Integer nroHabitacion) {

if (!contieneFechaIngreso(fechaIngreso) && !hayInternacionActiva()) {

area=area.toUpperCase();

if (area.equals("CARDIOLOGIA") || area.equals("GENERAL") || area.equals("PEDIATRIA")) {

Internacion internacion = new Internacion(importe, fechaIngreso, obraSocial, area, descuento,

nroHabitacion);

internaciones.addLast(internacion);

return true;

}

}

return false;

}

el metodo booleano agregarInternacion recibe por parameto area, una fecha de ingreso, double con un importe y un integer con el numero de habitacion, luego en el metodo se fija si contiene una fecha de ingreso y si no hay una intrnacion activa, luego transforma el area pasada por parametro a un area en mayusculas y luego verifica si el area pasada por parametro corresponde a alguna de esas, si cumple la conficion se crea una internacion y se la agrega y al final retorna true

si no retorna false

public boolean darAlta(Fecha fechaAlta) throws ParseException {

if (!internaciones.isEmpty()) {

Internacion ultimaInternacion = internaciones.getLast();

if (hayInternacionActiva() && ultimaInternacion.obtenerFechaIngreso().esMayorIgual(fechaAlta)) {

ultimaInternacion.agregarFechaAlta(fechaAlta);

// (porcentaje \* fechaIngreso \*costo diario)

Double costoAlta = descuento \* ultimaInternacion.obtenerFechaIngreso().obtenerDias(fechaAlta)

\* ultimaInternacion.obtenerImporte();

ultimaInternacion.agregarImporte(costoAlta);

saldo = saldo + costoAlta;

return true;

}

}

return false;

}

el metodo publico boolean daralta recibe por parametro una fecha de alta, luego se fija si la lista de internacionesno esta vacia si no lo esta instancia a la ultima internacion y luego verifica si hay internaciones activas y la fecha de alta es mayor a la fecha de ingreso, luego se agrega una fecha de alta a ultima internacion, luego se realiza en na variable costoalta la multiplicacion entre el descuento la internacion y el importe y luego al saldo se le suma el costo de alta y se returna true, en caso de no darse nnguna verificacion se retorna false

public void pagarSaldo() throws Exception {

if (!internaciones.isEmpty()) {

for (Internacion internacion : internaciones) {

if (internacion.obtenerFechaAlta() != null)

internacion.pagar();

}

saldo = 0.0;

}

else {

throw new Exception("No hay atenciones para pagar");

}

}

este metodo se fija si las internaciones no estan vacias y si no lo esta las recorre y si la internacion no esta dada de alta la paga y pone su saldo en 0 sino devuelve una excepcion

CLASE PACIENTE PRIVADO HEREDADO DE PACIENTE

private HashMap<Fecha, Atencion> atenciones;

posee un hashmap de tipo privado con una fecha y la atencion llamado atenciones

public PacientePrivado(String nombre, Integer nroHistoriaClinica, Fecha nacimiento) {

super(nombre, nroHistoriaClinica, nacimiento);

atenciones = new HashMap<Fecha, Atencion>();

}

constructor pacienteprivado que recibe por parametro un nombre, un numero de historia clinica, una fecha de nacimiento, este constructor tiene un super adentro que hereda de paciente y le ingresa un nombre, un numero de historia clinica y un nacimiento, luego inicializa un hashmap de atenciones

public boolean agregarAtencionConsultorio(Fecha fecha, Medico medico,Double importe) {

if (!atenciones.containsKey(fecha)) {

Consultorio consulta = new Consultorio(importe, fecha, medico.obtenerNombre(),

medico.obtenerEspecialidad());

atenciones.put(fecha, consulta);

saldo = saldo + consulta.obtenerImporte();

return true;

}

return false;

}

el metodo publico booleano agregarAtencionConsultorio que recibe por parametro una fecha, un medico y un importe, luego verifica si no existe las atenciones, luego crea una consulta, y agrega la consulta en atenciones, luego el saldo se modifica y se retorna true

si no se cumple las condiciones retorna false

public boolean agregarAtencionGuardia(Fecha fecha) {

if (!atenciones.containsKey(fecha)) {

Atencion atencion = new Atencion(0.0, fecha);

atenciones.put(fecha, atencion);

return true;

}

return false;

}

el metodo publico booleano agregarAtencionGuardia recibe por parametro una fecha, verifica si no existe la atencion en atenciones, luego se crea la atencion, y se la agrega, luego retorna true, si no se cumpla ninguna condicion retorna false

public Map<Fecha, String> obtenerAtencionesConsultorio() {

Map<Fecha, String> atencionesConsultorio = new HashMap<Fecha, String>();

for (Atencion atencion : atenciones.values()) {

if (validarClaseAtencion(atencion, "Consultorio")) {

Consultorio consultorio = (Consultorio) atencion;

atencionesConsultorio.put(consultorio.obtenerFecha(), consultorio.obtenerEspecialidad());

}

}

return atencionesConsultorio;

}

el motodo publico que retorna un map con una fecha y un string llamado obtener atenciones consultorio

crea un map de atenciones en consultorio con una fecha y un string, luego recorre las atencion y valida si alguna corresponde al consultrio si correciones se la instancia y luego se la agrega y al final retorna todas las atenciones del consultorio

private boolean validarClaseAtencion(Atencion atencion, String tipo) {

if (atencion.getClass().getSimpleName().equals(tipo)) {

return true;

} else {

return false;

}

}

el metodo privado booleando validarClaseAtencion recibe una atencion y un string tipo por parametro, uego verifica si la atencion corresponde al tipo pasado por parametro y si corresponde retorna true sino retorna falso

public void pagarSaldo() throws Exception {

if (!atenciones.isEmpty()) {

for (Atencion atencion : atenciones.values()) {

atencion.pagar();

}

saldo = 0.0;

}

else {

throw new Exception("No hay atenciones para pagar");

}

}

el metodo publico void pagarsaldo lo que hace es pagar el saldo de un usuario, luego verifica si las atenciones estan vacias si no lo estan recorre las atenciones y las paga, luego pone el saldo en 0.0 si no se cumple ninguna condificion devuelve una excepcion

irrep

CLASE ATENCION

//LA ATENCION NO PUEDE TENER UN IMPORTE MENOR A 0

EL ATRIBUTO PAGADO NO PUEDE INICIAR EN TRUE

EL IMPORTE DE LA ATENCION va a ser como maximo 100000 y minimo 0

-----------------------------------

CLASE CONSULTORIO

EL NOMBRE DEL MEDICO TIENE QUE TENER UN LENGTH MAYOR A 0 Y MENOR A 50

LA ESPECIALIDAD NO PUEDE TENER UN LENGHT MENOR A 0 Y MAYOR A 50

----------------------------------

CLASE INTERNACION

EL DESCUENTO NO PUEDE SER MENOR A 0 Y MENOR A 100

EL AREA NO PUEDE TENER UN LENGHT MENOR A 0 Y MAYOR A 50

LA OBRA SOCIAL NO PUEDE TENER UN LENGHT MENOR A 0 Y MAYOR A 50

EL NUMERO DE HABITACION NO MENOR A 0

LA FECHA DE ALTA NO PUEDE SER MENOR A LA FECHA DE INGRESO

-----------------------------------

CLASE MEDICO

LA MATRICULA TIENE QUE SER MAYOR A 0

EL LENGTH DEL NOMBRE DEL MEDICO TIENE QUE SER MAYOR A 0 Y MENOR A 50

LA MATRICULA TIENE QUE SER MAYOR A 0

EL HONARIO DEL MEDICO TIENE QUE SER MAYOR A 0 Y MENOR A 100000

--------------------------------------

CLASE FECHA

EL DIA NO PUEDE SER MENOR A 0 Y MAYOR A 32

EL MES NO PUEDE SER MENOR A 0 Y MAYOR A 13

EL AÑO NO PUEDE SER MENOR A 1920 Y MAYOR A 2201

-------------------------------------

CLASE PACIENTE

EL NUMERO DE HISTORIA CLINICA NO PUEDE SER MENOR A 1

EL NACIMIENTO NO PUEDE SER MENOR AL DIA 01,01,1920

EL LARGO DEL NOMBRE NO PUEDE SER MENOR A 1 Y MAYOR A 50

EL SALDO NO PUEDE SER MENOR A 0 NI MAYOR 1000000

--------------------------------------

CLASE PACIENTEAMBULATORIO

HEREDA EL IRREP DE PACIENTE

el nombre del tratamiento esta entre 0 y 50

CLASE PACIENTEPRIVADO

HEREDA EL IRREP DE PACIENTE

CLASE PACIENTE OBRA SOCIAL

HEREDA EL IRREP DE PACIENTE

el descuento no puede ser menor a 1 ni mayor a 100

la largo del nombre de obra social no puede ser menor a 1 ni mayor a 50

------------------------------------------

CLASE CENTRO MEDICO

el cuit tiene que tener un largo igual a 13

el nombre tiene que tener un largo maximo como 50 y un minimo como 1

el costo del dia de internacion no puede ser mayor a 100000 ni menor a 0

el hashmap de medicos no puede recibir un integer menor a 1 y tampoco puede recibir un medico que no tenga los atributos de medico

el hashmap de especialidades no puede recibir un string menor a 1 y tampoco mayor a 50, el precio de la especialidad no puede ser mayor a 100000 ni menor a 1

el hashmap de pacientes no puede tener un integer menor a 1, y el paciente tiene que tener los atributos de paciente

el hashmap de obrassociales tiene que tener un string con un nombre de largo mayor a 1 y menos a 50 y el descuento tiene que ser mayor a 0 y menor a 100

el hashmap de habitaciones tiene que tener un integer mayor a 0 y menor a 10

**Conclusión**

Concluimos que pasar de la especificación a la implementación a veces puede ser complicado ya que uno a veces piensa que pasar de la especificación a la implementación es fácil pero a veces puede complicarse un poco.

Crear una aplicación lleva tiempo y eso requiere de mucho esfuerzo y conocimiento, supimos que no se puede hacer un programa de mucha complejidad, de un dia para el otro, lleva mucho más tiempo.

Encontramos en la realización del trabajo práctico una experiencia realmente enriquecedora, ya que haciendo este programa, entendemos más a fondo la máquina y eso nos proporciona un nuevo conocimiento que nos sirve en un futuro.