Programación II - Trabajo Práctico Integrador 1er Cuatrimestre 2023 SEGUNDA PARTE

A la hora de realizar la implementación del código nos surgieron problemas, como por ejemplo en qué tipo de colecciones utilizar a la hora de almacenar datos para poder acceder a ellos de forma O(1), por esta razón decidimos utilizar HashMap.
Por otra parte utilizamos la HERENCIA, POLIMORFISMO,SOBREESCRITURA, a la hora de crear las clases de los servicios.
También utilizamos StringBuilder a la hora de generar el método toString en algunas clases donde el dato string se actualizaba constantemente.

Clase empresa De Servicios:

Atributos:

servicios la clave es el código de servicio y el valor el servicio.
facturacionPorServicio la clave es el string del servicio, valor facturación.
serviciosRealizadosPorTipo la clave es el string del servicio, valor la cantidad.
especialistasRegistrados la clave el el código de especialista, el valor el especialista.
clientesRegistrados la clave es el dni, valor el Cliente.

Utilizamos la Colección Map porque nos sirve a la hora de acceder a datos de forma O(1).

```
public class EmpresaDeServicios {
    private String nombreEmpresa;
    private Map<Integer, Servicio> servicios;
    private Map<String, Double> facturacionPorServicio;
    private Map<String, Integer> serviciosRealizadosPorTipo;
    private Map<Integer, Especialista> especialistasRegistrados;
    private Map<Integer, Cliente> clientesRegistrados;
    private double facturacionTotal;
```

Construcor:

Creamos el constructor y seteamos los servicios realizados por tipo porque nos daba inconvenientes a la hora de actualizar los servicios realizados por tipo.

```java
public EmpresaDeServicios(String nombre) {
    nombreEmpresa = nombre;
    servicios = new HashMap<Integer, Servicio>();
    facturacionPorServicio = new HashMap<String, Double>();
    serviciosRealizadosPorTipo = new HashMap<String, Integer>();
    setearServiciosRealizadosPorTipo();
    especialistasRegistrados = new HashMap<Integer, Especialista>();
    clientesRegistrados = new HashMap<Integer, Cliente>();
    facturacionTotal = 0;
}

private void setearServiciosRealizadosPorTipo() {
    serviciosRealizadosPorTipo.put("Pintura", 0);
    serviciosRealizadosPorTipo.put("PinturaEnAltura", 0);
    serviciosRealizadosPorTipo.put("Electricidad", 0);
    serviciosRealizadosPorTipo.put("GasistaInstalacion", 0);
    serviciosRealizadosPorTipo.put("GasistaRevision", 0);


}
```

Métodos:

Método registrar cliente:

```
ublic void registrarCliente(Integer dni, String nombre, String telefono)
   Cliente cliente = new Cliente(dni, nombre, telefono);
   if (clientesRegistrados.containsKey(dni)) {
       throw new RuntimeException("Dni ya registrado en el sistema");
   }
   clientesRegistrados.put(dni, cliente);
```

IREP

dni ingresado no debe estar registrado .

Método registrar especialista:

```
public void registrarEspecialista(Integer numEspecialista, String nombre, String telefono, String especialidad){
    if(!serviciosRealizadosPorTipo.containsKey(especialidad))
        throw new RuntimeException ("Especialidad desconocida");
    Especialista especialista = new Especialista(numEspecialista, nombre, telefono, especialidad);
    if (especialistaRegistrado(numEspecialista)) {
        throw new RuntimeException("NumeroEspecialista ya registrado");
    }
    especialistasRegistrados.put(numEspecialista, especialista);
}
```

IREP
serviciosRealizadosPorTipo.containsKey(especialidad)== true ,debe ser una especialidad registrada

El número de especialistas no debe estar registrado.

Metodo solicitar Servicio Electricidad:
Aplicamos herencia.

```
public int solicitarServicioElectricidad(int dni, int nroEspecialista, String direccion, double precioPorHora,
        int horasTrabajadas) {
    if (!clientesRegistrados.containsKey(dni)) {
        throw new RuntimeException("Dni no registrado");
    }
    if (!especialistaRegistrado(nroEspecialista)) {
        throw new RuntimeException("NroEspecialista no registrado");
    }

    if (!especialidadEspecialista(nroEspecialista, "Electricidad")) {
        throw new RuntimeException("El especialista no realiza dicha tarea");
    }

    if ((precioPorHora <= 0) || (horasTrabajadas <= 0)) {
        throw new RuntimeException("PrecioPorHora o HoraTrabajada son menor a 0");
    }
    Servicio electricidad = new Electricidad(dni, nroEspecialista, direccion, precioPorHora, horasTrabajadas);
    this.servicios.put(electricidad.getCodServicio(), electricidad);
    this.especialistasRegistrados.get(nroEspecialista).agregarCodigoServicio(electricidad.getCodServicio());
    return electricidad.getCodServicio();
}
```

IREP
dni debe estar registrado

número de especialista debe estar registrado

precio por hora y horas trabajadas no pueden ser <=0.

Metodo solicitar servicio Pintura:
Aplicamos herencia.

```
blic int solicitarServicioPintura(int dni, int nroEspecialista, String direccion, int metrosCuadrados,
     double precioPorMetroCuadrado){
 if (!clientesRegistrados.containsKey(dni)) {
     throw new RuntimeException("Dni no registrado");
 }
 if (!especialistaRegistrado(nroEspecialista)) {
     throw new RuntimeException("NroEspecialista no registrado");
 }
 if (!especialidadEspecialista(nroEspecialista, "Pintura")) {
     throw new RuntimeException("El especialista no realiza dicha tarea");
 }
 if ((precioPorMetroCuadrado <= 0) || (metrosCuadrados <= 0)) {
     throw new RuntimeException("PrecioPorMetroCuadrado o metrosCuadrados son menor a 0");
 }
 Servicio pintura = new Pintura(dni, nroEspecialista, direccion, metrosCuadrados, precioPorMetroCuadrad
 this.servicios.put(pintura.getCodServicio(), pintura);
 this.especialistasRegistrados.get(nroEspecialista).agregarCodigoServicio(pintura.getCodServicio());
 return pintura.getCodServicio();
```

IREP

dni debe estar registrado

número de especialista debe estar registrado

precio por metro cuadrado y metros cuadrados no pueden ser <=0.

Metodo solicitar servicio pintura en altura:
Aplicamos herencia.
En este método utilizamos el concepto de sobrecarga a la hora de llamar a solicitarServicioPintura.

```
public int solicitarServicioPintura(int dni, int nroEspecialista, String direccion, int metrosCuadrados,
        double precioPorMetroCuadrado, int piso, double seguro, double alqAndamios){
    if (!clientesRegistrados.containsKey(dni)) {
        throw new RuntimeException("Dni no registrado");
    }
    if (!especialistaRegistrado(nroEspecialista)) {
        throw new RuntimeException("NroEspecialista no registrado");
    }
    if (!especialidadEspecialista(nroEspecialista, "PinturaEnAltura")) {
        throw new RuntimeException("El especialista no realiza dicha tarea");
    }
    if ((precioPorMetroCuadrado <= 0) || (metrosCuadrados <= 0) || (piso <= 0) || (seguro <= 0)
            || (alqAndamios <= 0)) {
        throw new RuntimeException("Parametros <=0");
    }
    Servicio pinturaAltura = new PinturaAltura(dni, nroEspecialista, direccion, metrosCuadrados, precioPorMetroCuadrado, pis
    this.servicios.put(pinturaAltura.getCodServicio(), pinturaAltura);
    this.especialistasRegistrados.get(nroEspecialista).agregarCodigoServicio(pinturaAltura.getCodServicio());
    return pinturaAltura.getCodServicio();
}
```

IREP

dni del cliente debe estar registrado

número de especialista debe estar registrado

precio por metro cuadrado , metros cuadrados, piso y seguro no pueden ser <=0.

Metodo solicitar servicio de gasista instalación:
Aplicamos herencia.

```
public int solicitarServicioGasistaInstalacion(int dni, int nroEspecialista, String direccion, int cantArtefactos,
        double precioPorArtefacto){
    if (!clientesRegistrados.containsKey(dni)) {
        throw new RuntimeException("Dni no registrado");
    }
    if (!especialistaRegistrado(nroEspecialista)) {
        throw new RuntimeException("NroEspecialista no registrado");
    }
    if (!especialidadEspecialista(nroEspecialista, "GasistaInstalacion")) {
        throw new RuntimeException("El especialista no realiza dicha tarea");
    }
    if (precioPorArtefacto <= 0 || cantArtefactos <= 0) {
        throw new RuntimeException("Precio y cantidad de artefactos no pueden ser <=0 ");
    }

    Servicio gasistaInstalacion = new GasistaInstalacion(dni, nroEspecialista, direccion, cantArtefactos, precioPorArtefacto);
    this.servicios.put(gasistaInstalacion.getCodServicio(), gasistaInstalacion);
    this.especialistasRegistrados.get(nroEspecialista).agregarCodigoServicio(gasistaInstalacion.getCodServicio());
    return gasistaInstalacion.getCodServicio();
}
```

IREP

dni debe estar registrado

número de especialista debe estar registrado.
generamos una excepción

precio por artefacto y cantidad de artefactos no pueden ser <=0.

Metodo solicitar servicio gasista Revisión:
Aplicamos herencia.

```
ublic int solicitarServicioGasistaRevision(int dni, int nroEspecialista, String direccion, int cantArtefactos,
      double precioPorArtefacto){
  if (!clientesRegistrados.containsKey(dni)) {
      throw new RuntimeException("Dni no registrado");
  }
  if (!especialistaRegistrado(nroEspecialista)) {
      throw new RuntimeException("NroEspecialista no registrado");
  }
  if (!especialidadEspecialista(nroEspecialista, "GasistaRevision")) {
      throw new RuntimeException("El especialista no realiza dicha tarea");
  }
  if (precioPorArtefacto <= 0 || cantArtefactos <= 0) {
      throw new RuntimeException("Precio y cantidad de artefactos no pueden ser <=0 ");
  }
  Servicio gasistaRevision = new GasistaRevision(dni, nroEspecialista, direccion, cantArtefactos, precioPorArtefacto);
  this.servicios.put(gasistaRevision.getCodServicio(), gasistaRevision);
  this.especialistasRegistrados.get(nroEspecialista).agregarCodigoServicio(gasistaRevision.getCodServicio());
  return gasistaRevision.getCodServicio();
```

IREP

dni debe estar registrado

número de especialista debe estar registrado

precio por artefacto y cantidad de artefactos no pueden ser <=0.

Método finalizar servicio:
Para que este método responda en O(1) utilizamos un HasMap para poder acceder a el servicio a través de la clave que es el código de servicio.Una vez obtenido el servicio llama al método finalizarServicio de la clase SERVICIO para devolver el costo.
Luego le sumo el costo de materiales y devuelvo el total.

También sumamos la facturación por tipo de servicio a través de un método para que pueda responder en O(1)
utilizando un HashMap para acceder y poder incrementar los costos.

También sumamos la cantidad de servicios realizados por tipo a través de un método.

```
public double finalizarServicio(int codServicio, double costoMateriales) {
    if(!servicios.containsKey(codServicio))// O(1)
        throw new RuntimeException ("Codigo de servicio no existe");
    double costo = servicios.get(codServicio).finalizarServicio(costoMateriales); // 2 + O(1)
    String tipoServicio = servicios.get(codServicio).getClass().getSimpleName();// 2 + O(1)
    sumarCantidadDeServiciosRealizadosPorTipo(tipoServicio);// O(1)
    sumarfacturacionPorServicio(tipoServicio,costo);// O(1)
    int nroEspecialista = servicios.get(codServicio).getNroEspecialista();// 2 + O(1)
    especialistasRegistrados.get(nroEspecialista).actualizarHistorial(servicios.get(codServicio),codServicio);// O(1) + O (1)
    this.facturacionTotal += costo; // 2
    return costo;// 1
    //F(n) = O(1) + 2 + O(1) + 2 + O(1) + O(1) + O(1) + 2 + O(1) + O(1) + O (1) + O(1) + 2 + 1
    //F(n) = O(1) + 9 + O(1) + O(1) + O(1) + O(1) + O(1) + O(1) + O(1) + O(1)
    //F(n) =  O(1) + O(9) + O(1) + O(1) + O(1) + O(1) + O(1) + O(1) + O(1) + O(1)
    //F(n) = O(1)
}
```

IREP

servicios.containsKey(código servicio) == true.

Método sumar cantidad de servicios realizados por tipo:
Accedemos a la colección de tipo HashMap e incrementamos la cantidad.

```
private void sumarCantidadDeServiciosRealizadosPorTipo(String tipoServicio) {
    if (!serviciosRealizadosPorTipo.containsKey(tipoServicio))
        serviciosRealizadosPorTipo.put(tipoServicio, 1);
    else
        serviciosRealizadosPorTipo.put(tipoServicio, serviciosRealizadosPorTipo.get(tipoServicio) + 1);

}
```

Método sumar facturación por servicio:
Accedemos a la colección de tipo HashMap e incrementamos la cantidad.

```java
private void sumarfacturacionPorServicio(String tipoServicio, double costo) {
    if (!facturacionPorServicio.containsKey(tipoServicio))
        facturacionPorServicio.put(tipoServicio, costo);
    else
        facturacionPorServicio.put(tipoServicio, facturacionPorServicio.get(tipoServicio) + costo);
}
```

Metodo cantidad de servicios realizados por tipo:
En este método generamos una copia para de los servicios para no generar aliasing y poder proteger los datos.

```java
public Map<String, Integer> cantidadDeServiciosRealizadosPorTipo() {
    Map<String,Integer> nuevo = new HashMap<String,Integer>();
    nuevo.putAll(serviciosRealizadosPorTipo);
    return nuevo;// devolver un map distinto no generar aliasing
}
```

Método facturación total por tipo:
Este método responde en O(1) ya que utilizamos una colección de HashMap para poder acceder a la clave, donde tiene el valor de la facturación.

```java
public double facturacionTotalPorTipo(String tipoServicio) {
    return facturacionPorServicio.get(tipoServicio);// O(1)
    //F(n) = O(1).
}
```

Método facturación total:
Accedemos al atributo facturación total y lo devolvemos.

```java
public double facturacionTotal() {
    return facturacionTotal;
}
```

Metodo cambiar responsable:
Para que el método responda en O(1) utilizamos la colección HashMap que almacena los servicios para poder acceder a ellos y poder modificar el número de especialista que contiene.

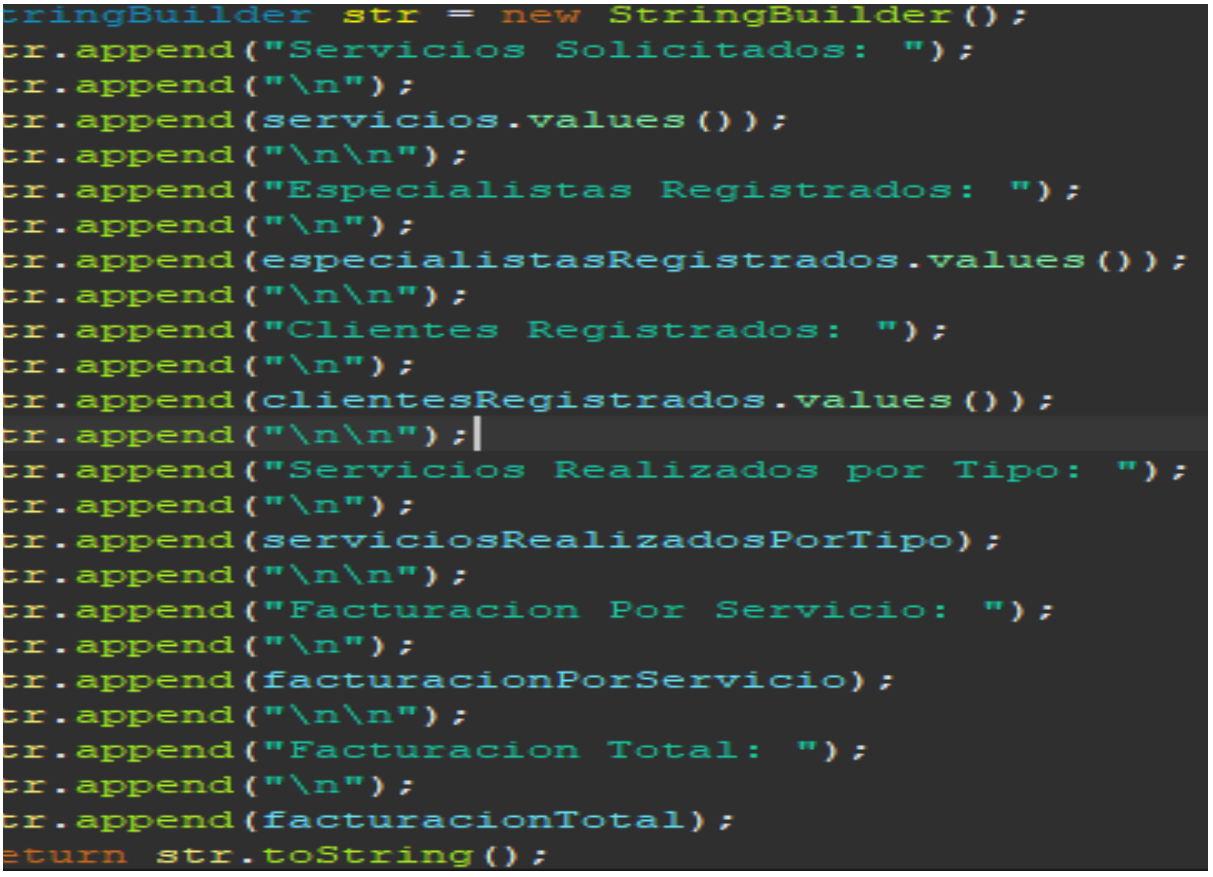
```java
public void cambiarResponsable(int codServicio, int nroEspecialista) {
    if(!codigoServicioRegistrado(codServicio))// O(1)
        throw new RuntimeException ("El codigo de servicio no esta Registrado");
    String especialidad = servicios.get(codServicio).getTipoServicio();// 2 + O(1)
    if(!especialidadEspecialista(nroEspecialista, especialidad))//O(1)
        throw new RuntimeException ("El especialista no realiza este oficio");
    servicios.get(codServicio).setNroEspecialista(nroEspecialista);// O(1)

    //F(n) = O(1) + 2 + O(1)+ O(1) + O(1).
    //F(n) = O(1) + O(1) + O(1) + O(1) + O(1).
    //F(n) = O(1).
}
```

Metodo listado de servicios realizados por especialista:
Accedemos a la colección de especialistas registrados y retornamos el toString del especialista.

```java
public String listadoServiciosAtendidosPorEspecialista(int nroEspecialista) {
    return especialistasRegistrados.get(nroEspecialista).toString();//O(1)
    // F (n) = O(1) ya que solo accedemos a traves de un HashMap.
}
```

Método toString:
Utilizamos StringBuilder ya que es una cadena que se va a ir modifican



```java
tringBuilder str = new StringBuilder();
tr.append("Servicios Solicitados: ");
tr.append("\n");
tr.append(servicios.values());
tr.append("\n\n");
tr.append("Especialistas Registrados: ");
tr.append("\n");
tr.append(especialistasRegistrados.values());
tr.append("\n\n");
tr.append("Clientes Registrados: ");
tr.append("\n");
tr.append(clientesRegistrados.values());
tr.append("\n\n");
tr.append("Servicios Realizados por Tipo: ");
tr.append("\n");
tr.append(serviciosRealizadosPorTipo);
tr.append("\n\n");
tr.append("Facturacion Por Servicio: ");
tr.append("\n");
tr.append(facturacionPorServicio);
tr.append("\n\n");
tr.append("Facturacion Total: ");
tr.append("\n");
tr.append(facturacionTotal);
eturn str.toString();
```

do constantemente.

Clase Especialista:

Atributos y Constructor:

En este punto decidimos agregarle una lista de servicios para que el especialista tenga un historial de los servicios realizados, aunque nos surgieron dudas quizás pueda mejorarse.

```java
public class Especialista {
    private int numEspecialista;
    private String nombre;
    private String telefono;
    private String especialidad;
    private int codServicio;
    private List<Servicio> historial;

    public Especialista(int numEspecialista, String nombre, String telefono, String especialidad) {
        this.numEspecialista = numEspecialista;
        this.nombre = nombre;
        this.telefono = telefono;
        this.especialidad = especialidad;
        historial = new ArrayList<Servicio>();
    }
```

Métodos:

Método actualizar Historial:
Agregamos al historial un nuevo servicio.
Esta parte también tuvimos inconvenientes podría estar mejor resuelta.

```java
public void actualizarHistorial(Servicio Servicio, int cod) {
    Servicio.setCodServicio(cod);
    historial.add(Servicio);
}
```

Método devolverHistorial:
Devuelve el historial de los servicios atendidos.

```java
public String devolverHistorial(){
    StringBuilder str = new StringBuilder();
    str.append(historial);
    return str.toString();
}
```

Metodo agregarCodigoServicio:
Agregar el código del servicio a realizar.

```java
public void agregarCodigoServicio(int codigo) {
    this.codServicio = codigo;
}
```

Clase Cliente:

Atributos y Constructor:

```java
public class Cliente{
    private int dni;
    private String nombre;
    private String telefono;

    public Cliente(Integer dni, String nombre, String telefono) {
        this.dni = dni;
        this.nombre = nombre;
        this.telefono = telefono;
    }
```

Métodos:

Método toString:
Utilizamos StringBuilder ya que la cadena se va a modificar constantemente.

```java
@Override
public String toString() {
    StringBuilder str = new StringBuilder();
    str.append(" dni= ");
    str.append(dni);
    str.append(" Nombre= ");
    str.append(nombre);
    str.append(" Telefono= ");
    str.append(telefono);
    return str.toString();
}
```

Clase Servicio:

Esta clase es abstracta ya que los demás servicios heredaran de ella sus atributos y métodos.

Atributos y Constructor:

```
//  private String tipoServicio;
    private int cliente;
    private int nroEspecialista;
    private String direccion;
    private  int codServicio=0;
    private static int iniciarCodigo =100;

    public Servicio(int cliente, int nroEspecialista, String direccion) {
        this.cliente = cliente;
        this.nroEspecialista = nroEspecialista;
        this.direccion = direccion;
        this.iniciarCodigo ++;
        this.codServicio = iniciarCodigo +1;
```

Métodos:

Método finalizar Servicio:
Es el método principal que van a heredar los demás servicios para poder ser finalizados.

```
protected abstract double finalizarServicio(double costoMateriales);
```

Método toString:
Utilizamos StringBuilder ya que la cadena se va a modificar constantemente.

```java
public String toString() {
    StringBuilder str = new StringBuilder();
    str.append("\n");
    str.append(" [");
    str.append(codServicio);
    str.append(" - ");
    str.append(this.getClass().getSimpleName());
    str.append(" ]");
    str.append(direccion);
    str.append("\n");

    return str.toString();
```

Clase Servicio Electricidad:

Es una subclase de Servicio ya que queremos que herede sus atributos y métodos.

Atributos y Constructor:

```java
public class ServicioElectricidad extends Servicio{
    private double precioPorHora;
    private int horasTrabajadas;

    public ServicioElectricidad(int cliente, int nroEspecialista, String direccion, double precioPorHora, int horasTrabajadas) {
        super(cliente, nroEspecialista, direccion);
        this.precioPorHora = precioPorHora;
        this.horasTrabajadas = horasTrabajadas;
    }
```

Métodos:

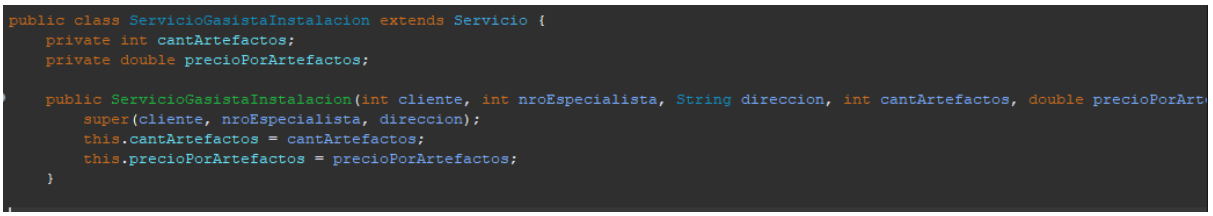
Método finalizar Servicio:
Sobreescribe el método heredado.

```java
erride
lic double finalizarServicio(double costoMaterial) {
 return costoMaterial + (precioPorHora * horasTrabajadas);
```

Clase Servicio Gasista Instalación:

Es una subclase de la clase Servicio.

Atributos y Constructor:

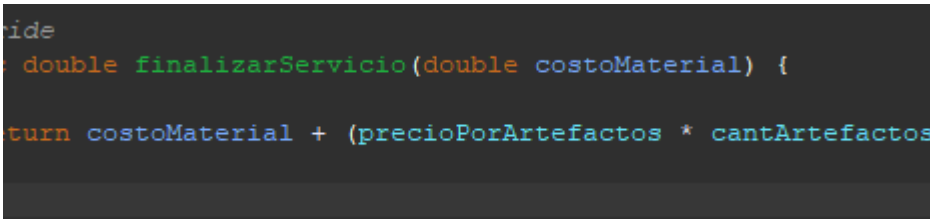


```
public class ServicioGasistaInstalacion extends Servicio {
    private int cantArtefactos;
    private double precioPorArtefactos;

    public ServicioGasistaInstalacion(int cliente, int nroEspecialista, String direccion, int cantArtefactos, double precioPorArt
        super(cliente, nroEspecialista, direccion);
        this.cantArtefactos = cantArtefactos;
        this.precioPorArtefactos = precioPorArtefactos;
    }
```

Métodos:

Método finalizar Servicio:
Sobreescribe el método heredado.



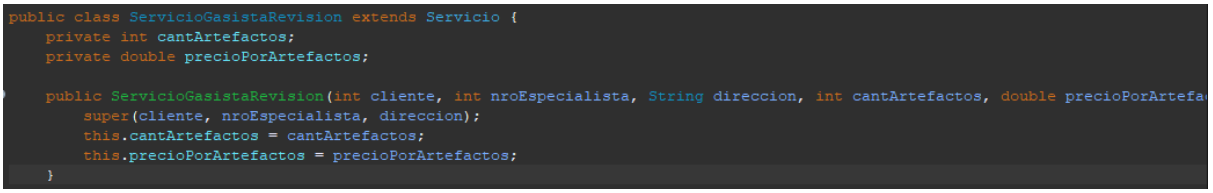
```
ide
 double finalizarServicio(double costoMaterial) {

turn costoMaterial + (precioPorArtefactos * cantArtefactos
```

Clase Servicio Gasista Revisión:

Es una subclase de la clase Servicio.

Atributos y Constructor:



```
public class ServicioGasistaRevision extends Servicio {
    private int cantArtefactos;
    private double precioPorArtefactos;

    public ServicioGasistaRevision(int cliente, int nroEspecialista, String direccion, int cantArtefactos, double precioPorArtefa
        super(cliente, nroEspecialista, direccion);
        this.cantArtefactos = cantArtefactos;
        this.precioPorArtefactos = precioPorArtefactos;
    }
```

Métodos:

Método finalizar Servicio:
Sobreescribe el método heredado.

```java
@Override
public double finalizarServicio(double costoMaterial) {
    double costoTotal = precioPorArtefactos * cantArtefactos;
    if(cantArtefactos>5)
        costoTotal = costoTotal - (costoTotal * 0.05);
    else if (cantArtefactos >10)
        costoTotal = costoTotal - (costoTotal * 0.15);
    return costoMaterial + costoTotal;
}
```

Clase Servicio Pintura:

Es una subclase de la clase Servicio.

```java
public class ServicioPintura extends Servicio {
    private int metrosCuadrados;
    private double precioPorMetroCuadrado;

    public ServicioPintura(int cliente, int nroEspecialista, String direccion, int metrosCuadrados,
            double precioPorMetroCuadrados) {
        super(cliente, nroEspecialista, direccion);
        this.metrosCuadrados = metrosCuadrados;
        this.precioPorMetroCuadrado = precioPorMetroCuadrados;
    }
```

Métodos:

Método finalizar Servicio:
Sobreescribe el método heredado.

```java
ide
 double finalizarServicio(double costoMaterial) {
turn costoMaterial +(precioPorMetroCuadrado * metrosCuadrad
```

Clase Servicio Pintura En Altura:

Es una subclase de la clase Pintura que a su ves hereda de servicio

```
public class ServicioPinturaAltura extends ServicioPintura{
    private int piso;
    private double seguro;
    private double alqAndamios;

    public ServicioPinturaAltura(int cliente, int nroEspecialista, String direccion, int metrosCuadrados,
            double precioPorMetrosCuadrados, int piso, double seguro, double alqAndamios) {
        super(cliente, nroEspecialista, direccion, metrosCuadrados, precioPorMetrosCuadrados);
        this.piso = piso;
        this.seguro = seguro;
        this.alqAndamios = alqAndamios;
    }
```

.

Métodos:

Método finalizar Servicio:
Sobreescribe el método heredado.

```
public double finalizarServicio(double costoMateriales) {
    if(piso>5)
        alqAndamios = alqAndamios + (alqAndamios * 0.20);
    double costoTotal = this.getMetrosCuadrados() * this.getPrecioPorMetroCuadrado();
     costoTotal  += + seguro + alqAndamios;
    return costoMateriales + costoTotal;
}
```

2)
Si quisiéramos acceder al historial de los clientes lo que tendríamos que hacer es iterar toda la colección de servicios y comparar el dni del cliente con el dni que almacena cada servicio en el atributo cliente, de esta forma podriamos devolver una colección nueva con los servicios solicitados por el cliente..
Al recorrer toda la colección completa la complejidad sería O(n) ya que estaremos comparando los dni en cada iteración de toda la colección.