TRABAJO PRÁCTICO INTEGRADOR PROGRAMACIÓN II

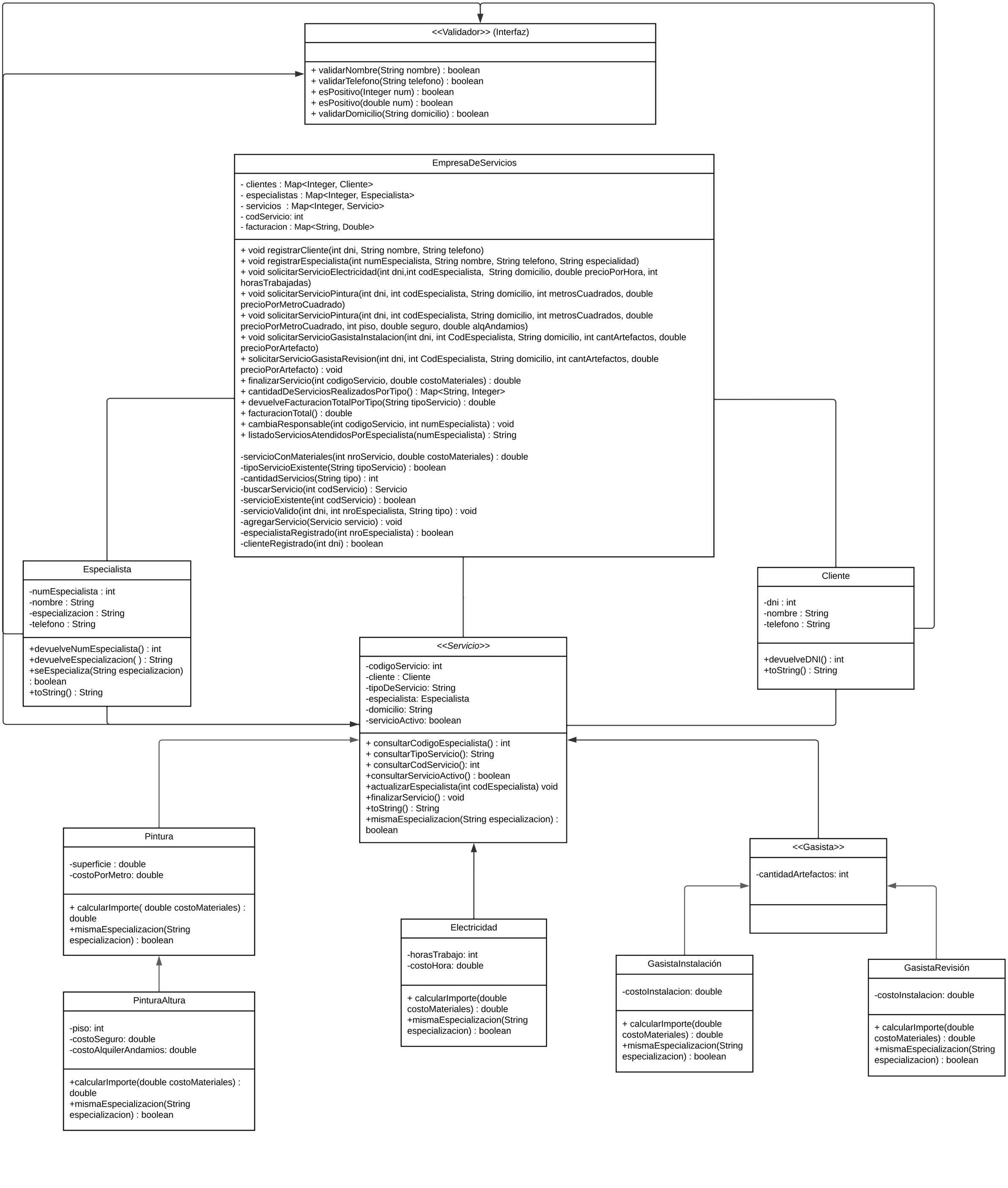
Nombre: Aquino Gaspar Abel

Legajo: 44.213.587

Comisión: 03

Objetivo del trabajo:

Utilizar todos los conceptos de programación aprendidos a lo largo de la cursada, tales como, Complejidad Computacional, TAD´S, Estructuras de Datos, Objetos, Herencia, Polimorfismo, Sobrecarga, Sobreescritura, Interfaz. Dichos conceptos deben ser utilizados para la especificación e implementación de un sistema de gestión de servicios para la empresa El Tano Construcciones.


<<Validador>> (Interfaz)
+ validarNombre(String nombre) : boolean
+ validarTelefono(String telefono) : boolean
+ esPositivo(Integer num) : boolean
+ esPositivo(double num) : boolean
+ validarDomicilio(String domicilio) : boolean
EmpresaDeServicios
- clientes : Map<Integer, Cliente>
- especialistas : Map<Integer, Especialista>
- servicios : Map<Integer, Servicio>
- codServicio: int
- facturacion : Map<String, Double>
+ void registrarCliente(int dni, String nombre, String telefono)
+ void registrarEspecialista(int numEspecialista, String nombre, String telefono, String especialidad)
+ void solicitarServicioElectricidad(int dni,int codEspecialista, String domicilio, double precioPorHora, int horasTrabajadas)
+ void solicitarServicioPintura(int dni, int codEspecialista, String domicilio, int metrosCuadrados, double precioPorMetroCuadrado)
+ void solicitarServicioPintura(int dni, int codEspecialista, String domicilio, int metrosCuadrados, double precioPorMetroCuadrado, int piso, double seguro, double alqAndamios)
+ void solicitarServicioGasistaInstalacion(int dni, int CodEspecialista, String domicilio, int cantArtefactos, double precioPorArtefacto)
+ solicitarServicioGasistaRevision(int dni, int CodEspecialista, String domicilio, int cantArtefactos, double precioPorArtefacto) : void
+ finalizarServicio(int codigoServicio, double costoMateriales) : double
+ cantidadDeServiciosRealizadosPorTipo() : Map<String, Integer>
+ devuelveFacturacionTotalPorTipo(String tipoServicio) : double
+ facturacionTotal() : double
+ cambiaResponsable(int codigoServicio, int numEspecialista) : void
+ listadoServiciosAtendidosPorEspecialista(numEspecialista) : String
-servicioConMateriales(int nroServicio, double costoMateriales) : double
-tipoServicioExistente(String tipoServicio) : boolean
-cantidadServicios(String tipo) : int
-buscarServicio(int codServicio) : Servicio
-servicioExistente(int codServicio) : boolean
-servicioValido(int dni, int nroEspecialista, String tipo) : void
-agregarServicio(Servicio servicio) : void
-especialistaRegistrado(int nroEspecialista) : boolean
-clienteRegistrado(int dni) : boolean
Especialista
-numEspecialista : int
-nombre : String
-especializacion : String
-telefono : String
+devuelveNumEspecialista() : int
+devuelveEspecializacion( ) : String
+seEspecializa(String especializacion) : boolean
+toString() : String
Cliente
-dni : int
-nombre : String
-telefono : String
+devuelveDNI() : int
+toString() : String
<<Servicio>>
-codigoServicio: int
-cliente : Cliente
-tipoDeServicio: String
-especialista: Especialista
-domicilio: String
-servicioActivo: boolean
+ consultarCodigoEspecialista() : int
+ consultarTipoServicio(): String
+ consultarCodServicio(): int
+consultarServicioActivo() : boolean
+actualizarEspecialista(int codEspecialista) void
+finalizarServicio() : void
+toString() : String
+mismaEspecializacion(String especializacion) : boolean
Pintura
-superficie : double
-costoPorMetro: double
+ calcularImporte(double costoMateriales) : double
+mismaEspecializacion(String especializacion) : boolean
PinturaAltura
-piso: int
-costoSeguro: double
-costoAlquilerAndamios: double
+calcularImporte(double costoMateriales) : double
+mismaEspecializacion(String especializacion) : boolean
Electricidad
-horasTrabajo: int
-costoHora: double
+ calcularImporte(double costoMateriales) : double
+mismaEspecializacion(String especializacion) : boolean
<<Gasista>>
-cantidadArtefactos: int
GasistaInstalación
-costoInstalacion: double
+ calcularImporte(double costoMateriales) : double
+mismaEspecializacion(String especializacion) : boolean
GasistaRevisión
-costoInstalacion: double
+ calcularImporte(double costoMateriales) : double
+mismaEspecializacion(String especializacion) : boolean

CONCEPTOS UTILIZADOS:

Herencia:

Para el ahorro de código opté por que las clases que sean un tipo de servicio hereden de la clase *Servicio* debido a que los servicios compartían atributos y métodos. Tales como *tipoDeServicio* o devolver algún atributo para usarlo en algún momento.

Polimorfismo:

Debido a que cada servicio tenía una forma distinta de calcular la tarifa a cobrar, por lo qué implementé un método *calcularImporte*(), el cual lo tienen todas las clases que heredan de *Servicio*, sin embargo, en el código cuando recorro la estructura de dato que guarda objetos de tipo *Servicio*, cada clase lo implementa a su manera.

Sobreescritura:

Como mencioné anteriormente, cada clase tiene su forma de *calcularImporte*(), por lo que al implementarlo de distintas maneras, se sobrescribe el método, es decir, se redefine. Esto también ocurre con las clases *Servicio*, *Cliente, Especialista* y *EmpresaDeServicios* con el método *toString().*

Interfaz:

Como debía tener en cuenta los IREP de las clases, tuve que implementar una serie de métodos para validar que, al crear los objetos, no se rompan, es por eso que opté por una interfaz llamada *Validador*, la cual lleva consigo métodos que son utilizados para validar información. En lugar de crear estos métodos en cada clase que los necesite, decidí que implementen esta interfaz y así, busco no repetir código.

Abstracción:

Al diseñar la herencia de las clases *Servicio* y *Gasista*, noté que estas clases nunca iban a ser instanciadas, sino que se utilizan para la creación de otra, por lo que decidí que fuese abstractas.

ESTRUCTURAS DE DATOS UTILIZADOS:

Map:

La estructura de datos que utilicé para guardar datos fue un *Mapa*, específicamente un *HashMap*. Opté por esta estructura debido al rápido acceso que tiene en sus elementos, siendo esta de complejidad O(1), por lo que satisfacía algunos requerimientos que necesitaba cumplir con la implementación del algoritmo.

Utilicé 4 estructuras, donde guardo objetos de tipo *Cliente, Especialista* y *Servicio*.

IREP DE LAS CLASES:

EmpresaDeServicios:

clientes → No puede tener clientes repetidos, por eso se usa como clave el DNI.

Especialistas →No puede tener especialistas repetidos, por eso se usa como clave el código de especialista.

Servicios → No debe tener servicios repetidos, por eso se usa como clave el código de cada servicio.

Facturación → No debe tener como clave un tipo de servicio que no ofrece la empresa.

Servicio:

codServicio →No debe ser menor que 0, ni debe ser el mismo que el de otro objeto.

dniCliente→Debe pertenecer a un cliente registrado.

codEspecialista→Debe pertenecer a un profesional de la empresa.

tipoServicio→Debe ser un tipo de servicio que ofrece la empresa (no puede ser null o vacio).

Domicilio→No debe ser vacio o nulo.

Clases heredadas de Servicio:

Sus variables de instancia no deben ser negativas o 0.

Cliente:

Dni→Debe ser un número mayor que 0.

Nombre/Telefono→No debe estar vacio o nulo.

Especialista:

numEspecialista→Debe ser mayor que 0 y no debe ser el mismo que el de otro objeto.

Especialización→ Debe ser un tipo de servicio que ofrezca la empresa.

Nombre/Telefono→No debe estar vacio o nulo.

Cruza de datos:

- No se puede finalizar un servicio que no esté activo.
- El costo de materiales pasados por parámetros en los métodos, debe ser mayor que 0.
- En los métodos que solicitan servicios, los parámetros que sean montos de dineros o cantidad de horas/artefactos/superficie, etc. Deben ser mayores que 0, esto para que no se corrompan los montos o de un monto totalmente a lo esperado.
- En los métodos calcularImporte() también debe recibir un costo mayor a 0.

COMPLEJIDAD:

A:

```
public String listadoServiciosAtendidosPorEspecialista(int nroEspecialista)
        if(!especialistaRegistrado(nroEspecialista))  // O(1)
                throw new RuntimeException("El especialista no está registrado");
        StringBuilder listado = new StringBuilder();  //O(1)
        for(Servicio servicios: this.servicios.values())  //O(s)
                if(servicios.consultarCodigoEspecialista() == (nroEspecialista)) //O(1)
                        listado.append(servicios.toString()); //O(1)
        return listado.toString(); //O(1)
```

La complejidad de este algortimo es de O(s)

Esto porque *s* representa la cantidad de servicios a recorrer.

O(1) + O(1) + O(1) + O(1) + O(1) + O(s) = O(s)

B:

Si quiero conocer el historial de servicios de un cliente en particular, basta con utilizar el DNI del cliente y comprobar en los servicios que servicio está registrado con el DNI del cliente en cuestión y, agregarlos a un String o estructura de dato. Sería muy similar la del especialista, por lo que tendría una complejidad O(n) donde n es la cantidad de servicios a recorrer.