//Modelado de ejecución automática (deberá existir un objeto singleton que contenga a todos los servicios y configurarle un crontab cada X tiempo)

class PlanificadorServicios {

*List*<*Servicio*> servicios

    PlanificadorServicios(){}

*void* nuevoServicioParaProcesar(*Servicio* *servicio*){

        servicios.add(servicio)

    }

*void* realizarProcesamiento(){

        servicios.foreach(servicio *->* servicio.validacionAutomatica())

    }

}

//Modelado de documentos

abstract class Documento {

*EstadoDocumento* estado = EstadoDocumento.NO\_REVISADO

    abstract *boolean* esCopiaDe(*Documento* otroDocumento)

*void* nuevoEstado(*EstadoDocumento* nuevoEstado){

        estado = nuevoEstado

    }

}

class Texto extends *Documento* {

*void* esCopiaDe(*Documento* *otroDocumento*){

        distanciaDeLevenshtein(otroDocumento)

        //ALMACENO EL RESULTADO y ACTUALIZO EL ESTADO

    }

}

class Programa extends *Documento* {

*void* esCopiaDe(*Documento* *otroDocumento*){

        deteccionDeClonesBasadoEnArbolesSintacticos(otroDocumento)

        //ALMACENO EL RESULTADO y ACTUALIZO EL ESTADO

    }

}

class Partitura extends *Documento* {

*void* esCopiaDe(*Documento* *otroDocumento*){

        acusticFingerprint(otroDocumento)

        //ALMACENO EL RESULTADO y ACTUALIZO EL ESTADO

    }

}

//Modelado de estado de los documentos

enum EstadoDocumento {

    NO\_REVISADO, ORIGINAL, DUDOSO, COPIA

}

//Modelado de servicios

abstract class Servicio {

*List*<*Documento*> documentosParaRevisar = new *ArrayList*<>();

*CalidadDeServicio* nivelDeCalidad

*void* ejecutar(){

        this.validacionAutomatica()

        nivelDeCalidad.validacionManual(this)

    }

    abstract *void* validacionAutomatica(){}

}

class BuscadorDePlagios extends *Servicio* {

*Documento* documentoOriginal

    BuscadorDePlagios(*CalidadDeServicio* *nivelCalidad*, *Documento* *documentoOriginal*, *List*<*Documento*> *posiblesPlagios*){

        this.documentoOriginal = documentoOriginal

        this.documentosParaRevisar = posiblesPlagios

        this.nivelDeCalidad = nivelCalidad

    }

    @*override*

*void* validacionAutomatica() {

        documentosParaRevisar.foreach(documento *->* documento.esCopiaDe(documentoOriginal))

    }

}

class BuscadorDeCopias extends *Servicio* {

    BuscadorDeCopias(*CalidadDeServicio* *nivelCalidad*, *List*<*Documento*> *documentos*){

        this.documentosParaRevisar = documentos

        this.nivelDeCalidad = nivelCalidad

    }

    @*override*

*void* validacionAutomatica() {

        this.productoCartesianoDocumentos().foreach((original, posibleCopia) *->* posibleCopia.esCopiaDe(original))

    }

*List*<*Documento*> productoCartesianoDocumentos() {

        documentosParaRevisar.productoCartesianoCon(documentos)

    }

}

//Modelado de niveles de calidad de servicio

interface CalidadDeServicio {

*void* validacionManual(*Servicio* *servicio*)

}

class Bronce implements *CalidadDeServicio* {

}

class Plata implements *CalidadDeServicio* {

*double* porcentajeValidacionSimple

*void* validacionManual(*Servicio* *servicio*) {

        //LLAMO AL MÉTODO CORRESPONDIENTE DEL REPOSITORIO DE REVISORES

    }

}

class Oro implements *CalidadDeServicio* {

    final *double* porcentajeValidacionSimple = 100

*double* porcentajeValidacionCruzada

*void* validacionManual(*Servicio* servicio) {

        //LLAMO AL MÉTODO CORRESPONDIENTE DEL REPOSITORIO DE REVISORES

    }

}

//Modelado de revisores

class RevisorIndependiente {

*String* email

*int* documentosQuePuedeRevisar

*int* documentosRevisados

*boolean* puedeRecibirRevisiones() {

        return documentosRevisados < documentosQuePuedeRevisar

    }

*void* enviarRevision(*Documento* *documento, Documento* *documento*){

        //Envío el mail correspondiente mediante el sistema de envio de mails

    }

}

class RepositorioRevisores {

*List*<*RevisorIndependiente*> revisores

*List*<*RevisorIndependiente*> revisoresDisponibles(){

        return revisores.filter(revisor *->* revisor.puedeRecibirRevisiones())

    }

*void* repartirServicios(*Servicio* *servicio*, *double* *proporsionSimple*, *double* *proporsionCruzada*){

        this.repartirServiciosEnProporcionSimple(servicio, proporsionSimple)

        this.repartirServiciosEnProporcionCruzada(servicio, proporsionCruzada)

    }

*void* repartirServiciosEnProporcionSimple(*Servicio* *servicio*, *double* *proporcion*){

        /\*

            Realizo un algoritmo que tome la proporción de documentos a revisar

            y a una cantidad igual de revisores (de no alcanzar, cumplo con el

            punto bonus y envío un mail a un administrativo) y asigne uno a

            cada uno

        \*/

    }

*void* repartirServiciosEnProporcionCruzada(*Servicio* *servicio*, *double* *proporcion*){

        /\*

            Realizo un algoritmo que tome la proporción de documentos a revisar

            y a todos los revisores. Para determinar cuantos se envían a cada

            revisor podemos utilizar otro algoritmo (por ejemplo: si hay

            6 documentos enviar 2 a todos los revisores).

        \*/

    }

}

class EnvioDeMails {

}

**Diagrama de clases**

<http://www.plantuml.com/plantuml/png/ZLJ1Rjim3BtxAuIUaY7P1sn0q2mEXW1B7IJRhWDZiHaBMN8foIiwxDydsOgTBccpWm61V7ea7vzyQnsmyvNAIWNMYkMZfzfiiI5h6AtudGaX6Ei08juXDrIIiIERJXDHiwaDMp9wHrMjy3IoOFyC4iRPdyIUU3jMh81KF5JmH1KiG3kI87ECg7WTxHgrzQ3GHbXv0R2q3jTTEdwrHnco9Us91z1etEPG3i6HDAMlKBimGN-CVSBUEeRIYQxXuyInum5eDy3GKW67kAO6LOuBKB6zJ7JDu0EMtW6FnkA3Q0p9NeLFtZvwb93ZHi61Z1Nuv53BryeLJ0VIe2uGB4nDSEb-m99soFTGeaYTvVY2lscZl0OFAjR1VglZeH8R7GZQpKQClSHr18Tf75p1jkf8u_SAHMsu33Xuw1RQMwYlNh2vJdU5x2r4ykVvslCaVLS1RIjpZeDhegC2t8EYP-3WGf1cDCZsDIp4z5ivPUkr0dRaFCFRA3O7Xker46fVYQvuiaBRxrtnOxlyljhDys8YYktgy-fk_cKYyczviGkHHR5PpJyEDHHekgSIvF5XQe-DEMeShHxQAD423mQ9CCPk-fIj2BNhHCZ-_OjCP-BctOr8WVFq-rbltIo5Gt0QVF_JXRjBtL2OS0saR7O41j9pHsHdJsCgpIyzdONoWtZP-Gly7yH9kZ-5yCiDPATwfi67uMvHIr-flm00>

**Explicación sobre almacenamiento de resultados y reporte:**

Podrían almacenarse en una base de datos. En nuestro sistema habría que añadir un repositorio de resultados o reportes de donde cada usuario podrá acceder a los documentos que este desee.

**1) Cargar un servicio con todos sus datos necesarios, de forma programática.**

nivelPlataCon50PorcientoManual = new Plata()

libroDeDDSOriginal = new Texto()

listaDeLibrosParecidos = [libro1, libro2, libro3]

new Servicio(nivelPlataCon50PorcientoManual, libroDeDDSOriginal, listaDeLibrosParecidos)

**2) Realizar el análisis automático de los documentos a ciertas horas del día (aún no definimos cuáles).**

Implementado en el sistema y explicado (clase PlanificadorServicios).

**3) Notificar a los revisores.**

Realizado en el repositorio de revisores. Se llama al método desde el servicio al hacer la validación manual.

**4) Consultar el estado de cada documento de un servicio.**

El documento podría tener un atributo contador que almacene las veces que el resultado dio original y otro atributo contador que almacene las veces que el resultado dio copia. Entonces cada vez que termina de realizarse el método ‘esCopiaDe(Documento otroDocumento)’ se llamaría al método ‘actualizarEstado()’ que realiza las comparaciones necesarias entre ambos contadores para determinar el nuevo estado (si todos son copias => COPIA, si todos son originales => ORIGINAL, si hay variados => DUDOSO).

**5) Consultar el precio de un servicio, el cual depende de la cantidad de documentos involucrados y la calidad de servicio.**

Podría establecerse un método ‘precioBase()’ en cada nivel de calidad. Luego, cada documento podría tener un atributo ‘precioBase’. De esta forma desde el servicio realizo la sumatoria de la lista de documentos y a ese resultado le sumo el precio base del nivel de calidad obteniendo así el precio deseado.