

PREGUNTAS DE TEORÍA:

PREGUNTA 1 (1 PUNTO) Contestad a las siguientes preguntas:

Suponed que tenéis un programa en el que se definen clases e interfaces cuyas definiciones se muestran incompletas a continuación:

```
public class A{...}  
public interface I1{...}  
public interface I2{...}  
public class B extends A implements I1{...}  
public class C extends B implements I2{...}  
public class D implements I2 ;
```

¿Son correctas las siguientes líneas de código? Justificadlo brevemente.

1.1) A c = new C() ;

1.2) I1 i1 = new B() ;

Ahora imaginad que en la clase A se define el siguiente método, cuya cabecera es:

```
public void metodo1(B arg1, I1 arg2) ;
```

1.3) ¿Es correcto el siguiente código? Justificadlo brevemente.

```
A a = new A() ;  
B b = new B() ;  
C c = new C() ;  
a.metodo1(c,b) ;
```

1.4) ¿Es correcto el siguiente código? Justificadlo brevemente.

```
A a = new A() ;  
B b = new B() ;  
D d = new D() ;  
a.metodo1(b,d) ;
```

1.1) Cierto, puesto que C es subclase de B, que a su vez es subclase de A.

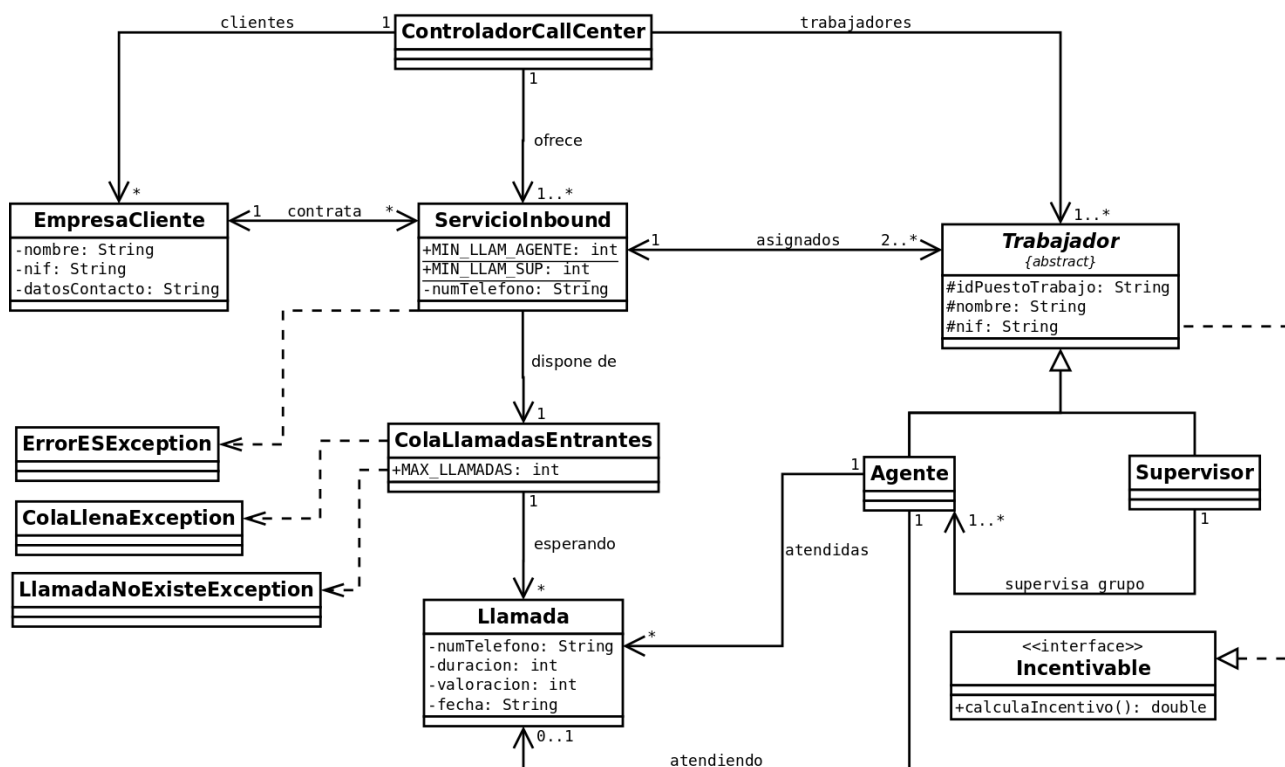
1.2) Cierto, puesto que B implementa la interface I1.

1.3) Cierto, puesto que C es subclase de B y B implementa la interface I1.

1.4) Falso, puesto que D no implementa la interface I1.

PREGUNTAS DE CODIFICACIÓN:

El diagrama de clases UML (incompleto en términos de métodos de las clases) que se muestra en la figura que sigue pretende ser el correspondiente al núcleo de una aplicación de gestión de llamadas entrantes en un centro de atención telefónica de clientes ("call center" en inglés) que puede ser subcontratado por diferentes empresas clientes (un ejemplo: la subcontratación por parte de una operadora telefónica de este tipo de asistencia para sus clientes).



Cuando una empresa subcontrata el servicio, se le asigna un teléfono. El servicio se representa con una instancia de la clase `ServicioInbound`. El atributo `numTelefono` de dicha clase contendrá el valor del número de teléfono (único) asignado al servicio subcontratado. Una misma empresa cliente puede subcontratar varios servicios (p.e., asistencia técnica, comercial, etc..., a sus propios clientes).

Cada servicio gestiona una cola de llamadas entrantes con una capacidad máxima definida por el valor de la constante `MAX_LLAMADAS` (p.e., igual a 25), atributo de la clase `ColaLlamadasEntrantes`. Una nueva llamada entrante se coloca en la última posición y se atiende primero la primera llamada de la cola.

A cada servicio se asigna un equipo de trabajo, compuesto por un supervisor y como mínimo un agente (persona encargada de responder a las llamadas entrantes). A cada trabajador de la empresa se le asigna un identificador único (atributo `idPuestoTrabajo`).

Cuando uno de los agentes asignados a un servicio finaliza la llamada que está atendiendo, el servicio captura la información de la llamada finalizada y la añade al contenedor de llamadas atendidas de dicho agente. Acto seguido, toma la primera llamada de la cola de llamadas entrantes y se la asigna para que la sirva (la asigna al atributo `atendiendo` de la instancia de `Agente`). **Este contenedor de llamadas atendidas se vacía cuando comienza un nuevo mes de forma automática.**

La aplicación guarda la siguiente información de las llamadas: el teléfono desde el que se realizó la llamada, su duración en segundos, su valoración por el cliente (un valor entre 1 y 5), y la fecha en que se realizó.

La empresa calcula a final de mes incentivos para sus trabajadores. Para los agentes, el incentivo toma el valor del 5% de su salario bruto mensual si ha atendido un mínimo de llamadas igual al valor de la constante `MIN_LLAM_AGENTE` de la clase `ServicioInBound` (p.e., igual a 600) y la media aritmética de su valoración es mayor que 3 y menor que 4; en caso que la media aritmética de su valoración es mayor o igual que 4, el incentivo tomará el valor del 6% de su salario. Para los supervisores, el incentivo toma el valor del 5% de su salario bruto mensual si sus agentes han atendido (entre todos ellos) un mínimo de llamadas igual al valor de la constante `MIN_LLAM_SUP` de la clase `ServicioInBound` (p.e., igual a 1800) y la media aritmética de su valoración es superior a 3,5.

PREGUNTA 2 (1,5 PUNTOS) A partir del diagrama de clases UML de la figura, escribid para las clases ControladorCallCenter, EmpresaCliente, ServicioInbound, Trabajador, Agente, Supervisor, ColaLlamadasEntrantes y Llamada la parte de código de la definición que comienza con el “public class...” correspondiente a la declaración de los todos los atributos de las clases, incluyendo aquellos que aparecen al implementar las relaciones mostradas en el diagrama. **NO debéis añadir nada más en el código de estas clases** en vuestras respuestas a esta pregunta.

```
public class ControladorCallCenter {
    private Map<String, EmpresaCliente> empresasClientes; //Clave: nif
    private Map<String, ServicioInBound> servicios; //Clave: telefono
    private Map<String, Trabajador> trabajadores; //Clave: idPuestoTrabajo
}

public class EmpresaCliente {
    private String nombre, nif, datosContacto;
    private Map<String, ServicioInBound> contratados; //Clave: telefono
}

public class ServicioInBound {
    public static final int MIN_LLAM_AGENTE = 600;
    public static final int MIN_LLAM_SUP = 1800;
    private String numTelefono;
    private EmpresaCliente empresa;
    private Map<String, Trabajador> grupoTrabajo; //Clave: idPuestoTrabajo
    private ColaLlamadasEntrantes cola;
}

public abstract class Trabajador implements Incentivable {
    protected String idPuestoTrabajo, nombre, nif;
    protected ServicioInBound servicio;
}

public class Agente extends Trabajador {
    private List<Llamada> atendidas;
    private Llamada atendiendo;
}

public class Supervisor extends Trabajador {
    private List<Agente> supervisados;
}

public class ColaLlamadasEntrantes {
    public static final int MAX_LLAMADAS = 25;
    private List<Llamada> cola;
}

public class Llamada {
    private String numTelefono, fecha;
    private int duracion, valoracion;
}
```

PREGUNTA 3 (0,75 PUNTO) Proponed y diseñad un constructor para la clase Trabajador. Proponed y

diseñad también un constructor para la clase Supervisor.

```
public abstract class Trabajador implements Incentivable {

    public Trabajador (String idPuestoTrabajo, String nombre, String nif) {
        this.idPuestoTrabajo = idPuestoTrabajo;
        this.nombre = nombre;
        this.nif = nif;
        this.servicio = null; //Aún no asignado cuando se crea el trabajador
    }
}

public class Supervisor extends Trabajador {

    public Supervisor (String idPuestoTrabajo, String nombre, String nif) {
        super (idPuestoTrabajo, nombre, nif);
        this.supervisados = new ArrayList<Agente>(); //Inicialmente vacía
    }
}
```

PREGUNTA 4 (0,75 PUNTOS) Implementad, en la clase ColaLlamadasEntrantes los siguiente métodos:

```
public void addLlamada(Llamada llamada) throws ColaLlenaException;
```

Este método intenta añadir un objeto llamada a la cola de llamadas entrantes. Si la cola no está llena la añade. En caso contrario lanza una excepción instancia de ColaLlenaException.

```
public Llamada extraeSiguiente() throws LlamadaNoExisteException;
```

Este método intenta extraer la primera llamada de la cola de llamadas entrantes. Si la no está vacía la elimina de la cola y la devuelve como resultado. En caso contrario lanza una excepción instancia de LlamadaNoExisteException.

```
public class ColaLlamadasEntrantes {

    public void addLlamada (Llamada llamada) throws ColaLlenaException {
        if (this.cola.size() == ColaLlamadasEntrantes.MAX_LLAMADAS) {
            throw new ColaLlenaException();
        }
        this.cola.add(llamada);
    }

    public Llamada extraeSiguiente() throws LlamadaNoExisteException {
        if (this.cola.isEmpty()) {
            throw new LlamadaNoExisteException();
        }
        return this.cola.remove(0);
    }
}
```

PREGUNTA 5 (1 PUNTO) Implementad, en la clase ColaLlamadasEntrantes, el siguiente método:

```
public void eliminaInterrumpida(String nTelefono) throws
```

LlamadaNoExisteException;

Este método intenta eliminar de la cola aquella llamada cuyo número de teléfono coincida con el valor del argumento `nTelefono`. Si la cola contiene tal llamada, ésta se elimina de la cola. En caso contrario lanza una excepción instancia de `LlamadaNoExisteException`.

```
public class ColaLlamadasEntrantes {

    public void eliminaInterrumpida (String nTelefono) throws
LlamadaNoExisteException {
        Llamada aEliminar = null;
        Iterator<Llamada> it = this.cola.iterator();
        while (it.hasNext() && aEliminar == null) {
            Llamada llamada = it.next();
            if (llamada.getNumTelefono().equals(nTelefono)) {
                aEliminar = llamada;
            }
        }

        if (aEliminar != null) {
            this.cola.remove(aEliminar);
        }
        else {
            throw new LlamadaNoExisteException();
        }
    }
}
```

PREGUNTA 6 (1 PUNTO) Mientras un agente está atendiendo una llamada, también ésta puede interrumpirse por diversos motivos. Si eso sucede, el atributo `atendiendo` del objeto `Agente` correspondiente, debe ponerse a `null`. Implementad, en la clase `ServicioInBound`, el siguiente método:

```
public void eliminaInterrumpida(String nTelefono);
```

Este método se invocará cuando, o bien se interrumpe una llamada de la cola en espera o bien se interrumpe una llamada que está siendo atendida por un agente.

Este método debe primero intentar eliminar de la cola aquella llamada cuyo número de teléfono coincida con el valor del argumento `nTelefono`. Si la cola contiene tal llamada, ésta se elimina de la cola. En caso contrario busca entre todos los agentes aquél que haya estado atendiendo una llamada desde ese número y pone su atributo `atendiendo` a `null`. Observad que este método NO lanza ninguna excepción, pues solo se invoca si efectivamente se interrumpe una llamada en espera o que está siendo atendida por un agente.

Podéis asumir que la clase contiene el método siguiente (**no debéis codificarlo**):

```
public List<Agente> getAgentes();
```

Este método devuelve una lista con todos los agentes que prestan ese servicio.

```
public class ServicioInBound {

    public void eliminaInterrumpida (String nTelefono) {
        try {
```

```

        this.cola.eliminaInterrumpida(nTelefono);
    }
    catch (LlamadaNoExisteException e) {
        List<Agente> agentes = this.getAgentes();
        boolean eliminada = false;
        Iterator<Agente> it = agentes.iterator();
        while (it.hasNext() && eliminada == false) {
            Agente a = it.next();
            Llamada atendiendo = a.getAtendiendo();
            if (atendiendo != null &&
                atendiendo.getNumTelefono().equals(nTelefono)) {
                a.setAtendiendo(null);
                eliminada = true;
            }
        }
    }
}

```

PREGUNTA 7 (1,5 PUNTOS) Implementad los siguientes métodos:

En la clase Agente, el método:

```
public double getMediaDeValoracion();
```

Este método debe devolver la media aritmética de las valoraciones obtenidas en las llamadas atendidas.

En la clase Supervisor, el método:

```
public double calculaIncentivo();
```

Este método debe devolver el porcentaje del incentivo correspondiente al supervisor en cuestión (0 o 5%) siguiendo las reglas definidas en el enunciado anteriormente.

```

public class Agente extends Trabajador {

    public double getMediaDeValoracion() {
        if (this.atendidas.isEmpty()) {
            return 0.0;
        }
        else {
            int valTotal = 0;
            for (Llamada llamada : this.atendidas) {
                valTotal += llamada.getValoracion();
            }
            return (double)valTotal/this.atendidas.size();
        }
    }
}

public class Supervisor extends Trabajador {

    public double calculaIncentivo() {
        int totalLlamadas = 0;
        double sumaMedias = 0.0;
    }
}

```

```

        for (Agente a : this.supervisados) {
            totalLlamadas += a.getNumeroAtendidas();
            sumaMedias += a.getMediaDeValoracion();
        }

        if (totalLlamadas >= ServicioInBound.MIN_LLAM_SUP && sumaMedias /
            this.supervisados.size() > 3.5)
            return 5.0;
        else
            return 0.0;
    }
}

```

PREGUNTA 8 (1 PUNTO) Implementad, en la clase `ServicioInBound` el siguiente método:

```
public void trabajadoresAFichero(String nombreF) throws ErrorESEException;
```

Este método debe generar un archivo presentando información de todos los trabajadores asignados al servicio. Cada línea contendrá dos datos separados por una coma: el identificador del trabajador y su incentivo.

El argumento `nombreF` contiene el nombre completo del archivo a crear. En caso de que se produzca alguna excepción de entrada/salida, el método la captura y genera y lanza una excepción instancia de `ErrorESEException`, con un mensaje idéntico al mensaje de la excepción original capturada.

```

public class ServicioInBound {

    public void trabajadoresAFichero (String nombreF) throws ErrorESEException {

        try {
            FileOutputStream fos = new FileOutputStream (nombreF, true);
            PrintWriter pw = new PrintWriter (fos);

            for (Trabajador t : this.grupoTrabajo.values()) {
                pw.println(t.getIdPuestoTrabajo() + "," + t.calculaIncentivo());
            }

            pw.close(); //Cerramos stream de jerarquía superior
        }
        catch (IOException e) {
            throw new ErrorESEException(e.getMessage());
        }
    }
}

```

PREGUNTA 9 (1,5 PUNTOS) Implementad, en la clase `ServicioInBound` los siguientes métodos:

```
public double mediaDeValoracionMaximaDeAgentes();
```

Este método debe encontrar la máxima media aritmética de valoración de los agentes asignados al servicio. Recordad que el método `getMediaDeValoracion()` de la clase `Agente` devuelve la media aritmética de valoración de un agente.

```
public List<Agente> agentesMasValorados();
```

Este método debe devolver una lista con los agentes que hayan obtenido la máxima media aritmética de valoración de los agentes asignados al servicio.

```
public class ServicioInBound {

    public double mediaDeValoracionMaximaDeAgentes() {

        double mediaMax = 0.0;
        List<Agente> agentes = this.getAgentes();
        for (Agente a : agentes) {
            double media = a.getMediaDeValoracion();
            if (media > mediaMax)
                mediaMax = media;
        }

        return mediaMax;
    }

    public List<Agente> agentesMasValorados() {

        double mediaMax = this.mediaDeValoracionesMaximaDeAgentes();
        List<Agente> masValorados = new ArrayList<Agente>();

        List<Agente> agentes = this.getAgentes();
        for (Agente a : agentes) {
            if (a.getMediaDeValoracion() == mediaMax)
                masValorados.add(a);
        }

        return masValorados;
    }
}
```