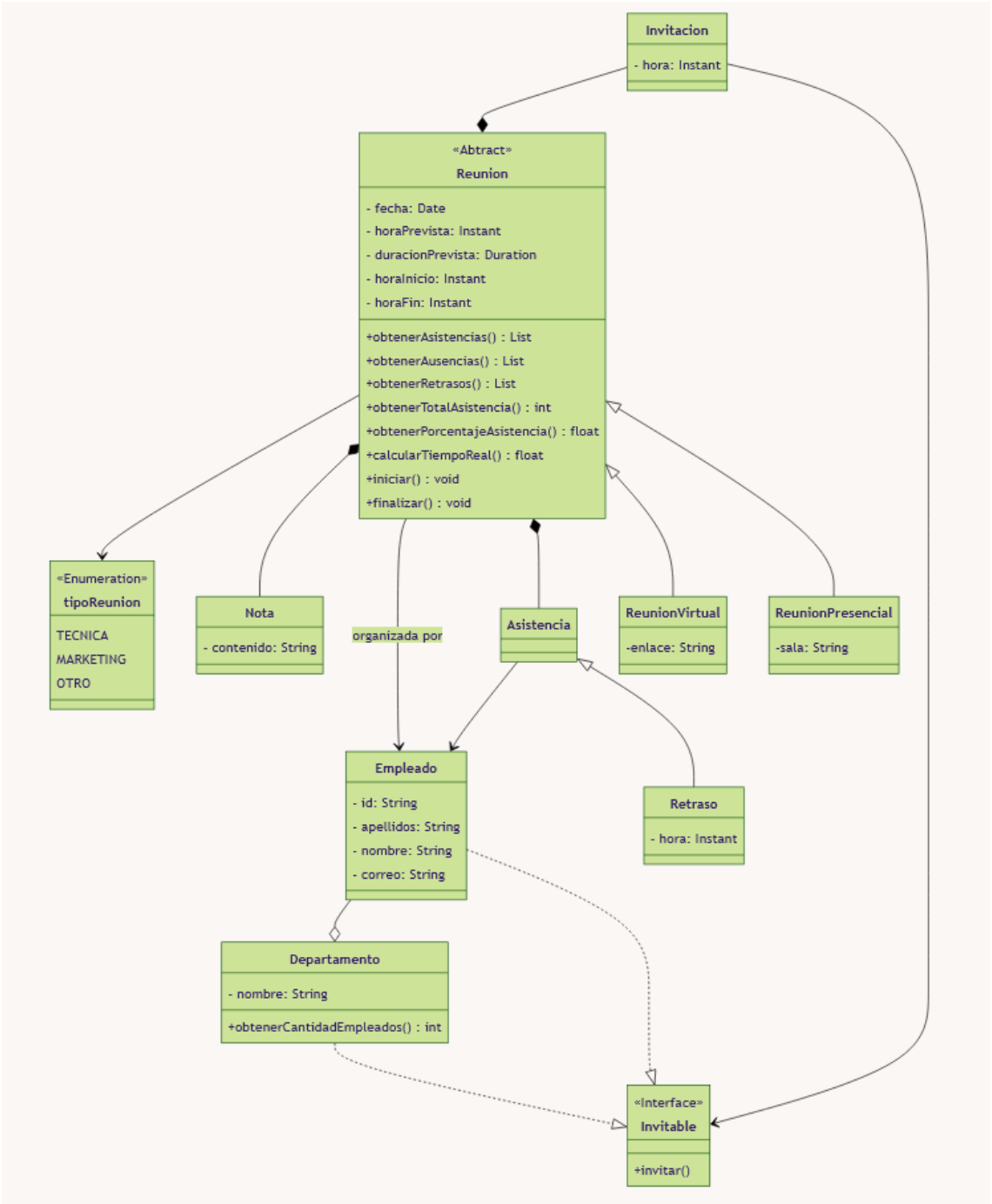
Enunciado Tarea 2

El objetivo de esta tarea es escribir el código del modelo UML visto en el último curso utilizando GIT. Esta tarea debe realizarse en grupos de **dos-tres personas.**

A modo de recordatorio, aquí está el texto y el modelo UML del ejercicio realizado durante el curso.

> Buscamos hacer un sistema que pueda gestionar las reuniones y, en particular, que nos permita saber quién asiste a cada reunión. Las reuniones pueden ser virtuales (con un enlace) o presenciales (en una sala).
>
> Cada reunión tiene una fecha, hora, duración prevista y lista de invitación (con sus horas). Cada reunión debe tener un organizador. Tenemos 3 tipos principales de reuniones: técnicas, de marketing y otros. Queremos poder iniciar y finalizar una reunión para calcular el tiempo real de la misma.
>
> Los invitados son nuestros empleados, que se identifican con un ID, tiene apellidos y nombre y correo. Cada empleado pertenece a un departamento específico de la empresa. Cada departamento tiene su propio nombre y queremos saber el número de empleados de cada departamento. También se puede invitar todo un departamento.
>
> Queremos poder añadir notas a una reunión.
>
> Para cada reunión, queremos tener una lista de quién está presente, quién está ausente y quién llega tarde (hacemos la lista al principio de la reunión, pero a menudo hay retrasos que de todas formas cuenta como asistencia). también queremos saber la hora de llegada de los que llegan tarde. También queremos poder obtener el número de participantes y el porcentaje de asistentes.


Invitacion
- hora: Instant
«Abtract»
Reunion
- fecha: Date
- horaPrevista: Instant
- duracionPrevista: Duration
- horaInicio: Instant
- horaFin: Instant
+obtenerAsistencias() : List
+obtenerAusencias() : List
+obtenerRetrasos() : List
+obtenerTotalAsistencia() : int
+obtenerPorcentajeAsistencia() : float
+calcularTiempoReal() : float
+iniciar() : void
+finalizar() : void
«Enumeration»
tipoReunion
TECNICA
MARKETING
OTRO
Nota
- contenido: String
organizada por
Asistencia
ReunionVirtual
-enlace: String
ReunionPresencial
-sala: String
Empleado
- id: String
- apellidos: String
- nombre: String
- correo: String
Retraso
- hora: Instant
Departamento
- nombre: String
+obtenerCantidadEmpleados() : int
«Interface»
Invitable
+invitar()

Deben escribir la implementación completa de cada una de las clases y las relaciones entre ellas.

Además de los métodos presentados aquí, debes implementar **los métodos getters y setters y el método toString de cada clase**. Este método toString debe mostrar una descripción relevante para la clase en cuestión tenga en cuenta que, para las clases compuestas, es posible hacer un loop y utilizar el toString de otras clases. En el futuro, las invitaciones se enviarán por correo electrónico conectándose al servidor de la empresa, pero de momento basta con imprimirlas desde la consola.

También puede modificar el diagrama UML (es muy posible que durante la aplicación encuentre mejoras que introducir), pero las modificaciones deben ser pertinentes y explicadas en el readme.

Pueden utilizar todas las clases estándar de Java para su implementación (String, ArrayList, Time...). No olviden utilizar la documentación de Java.

Además de todo esto, hay que añadir nuevas funciones al software :

1. La **posibilidad de elaborar un informe de la reunión**. De usted depende elegir los métodos y clases adecuados (pueden ser nuevos o cambios de lo que ya existe, pero es necesario **actualizar el UML**). El informe debe proporcionar al menos:
    - Fecha y hora de la reunión
    - Horas de inicio y fin, y duración total
    - Información sobre el tipo de reunión
    - Enlace o sala
    - Lista de participantes y información sobre retrasos
    - Todas las notas relacionadas con la reunión (ordenadas cronológicamente)

    Este informe debe almacenarse en un archivo txt. Puede ver este tutorial para ayudarle a escribir en un archivo.

2. Permitir la invitación y asistencia a la reunión de **invitados externos**. Aunque estas personas no sean empleados, el sistema debe permitirnos conocer sus nombres completos y una dirección de correo electrónico.

También deben escribir **tests unitarios que prueban toda la lógica de su aplicación.** Por ejemplo, crear e inscribirse en una reunión, añadir notas, iniciar y finalizar una reunión, generar informe, etc.

Su tests deben ser ejecutable (hacer un **proyecto Maven** con las dependencias de Junit).

Debe proporcionar un readme (en git) indicando los nombres completos de los estudiantes y el UML de su solución. **Deben explicar y justificar sus modelo con las nuevas funciones y los cambios que han realizado en el modelo en el archivo readme del proyecto.**

No es necesario hacer un main (pero aún es posible si le ayuda) sabiendo que las pruebas unitarias deben mostrar el comportamiento de su aplicación.

Documentación:

- Tienen que documentar el código usando el formato Javadoc que vimos durante el curso.
- No olvide que su editor de código (IntelliJ) puede ayudarle a generar la documentación

Pruebas:

- Deben usar Junit en su proyecto debe tener la configuración correcta para la ejecución de pruebas
- Sus pruebas deben cubrir casos de uso normales y verificar su correcto funcionamiento
- También debe probar los casos extremos, los casos que lanzan excepciones (Puede crear sus propias excepciones) o potencialmente con datos incorrectos o faltantes cuando sea relevante.
- Te recomiendo que escribas tantas clases de prueba como sea posible (no sólo una), siempre que haya lógica que probar (métodos que no sean simples getters o setters o constructores) entonces probablemente sea apropiado hacer varias clases de pruebas.
- Cada método de prueba debe ser breve y probar un escenario específico, no debe hacer afirmaciones sobre casos totalmente diferentes (por ejemplo, no pruebe todas las excepciones en un método de prueba).
- Sus pruebas deben ser coherentes con sus métodos y clases, no pruebe escenarios que no tengan nada que ver con el software.
- Puede crear pruebas más completas que comprueben una secuencia de eventos cuando sea necesario (para la generación del formulario, por ejemplo).

Readme y modelo UML:

- Debe proporcionar un readme (en git) indicando los nombres completos de los estudiantes
- Pueden incrustar su modelo UML actualizado en este readme o en una imagen en su repositorio.
- Deben explicar y justificar brevemente en el readme **cualquier adición o modificación al modelo UML**. En particular, hay que explicar cómo se gestionan las nuevas funciones y por qué se hace así.

La fecha límite para la rendición de la tarea en Canvas (el enlace de su código en github) es el viernes 23 de mayo a las 23:59.

Resultados de aprendizaje evaluados:

1. Aplicar el paradigma de la orientación a objetos al desarrollo de aplicaciones software de mediana envergadura, en un entorno de trabajo en equipos.
2. Comparar los beneficios de la orientación a objetos frente a otros paradigmas para el desarrollo de sistemas.

3. Aplicar principios de diseño para el desarrollo de sistemas informáticos de calidad.

5. Aplicar técnicas de testing unitario para pruebas de verificación de la funcionalidad que implementen.

Rúbrica de evaluación Tarea 2

Criterios	Calificaciones			Pts
Completitud de la aplicación KPI: 1.1	**12 para >8.0 pts** **Excelente** La aplicación contiene todo el código necesario y funciona correctamente siguiendo las instrucciones dadas.	**8 para >4.0 pts** **Bien** Faltan algunas partes menores del código o hay problemas menores de implementación.	**4 para >0 pts** **Por mejorar** Faltan algunas partes importantes del código o hay problemas importantes con la implementación.	12 pts
Calidad del código KPI: 1.3, 2.1	**6 para >4.0 pts** **Excelente** El código proporcionado utiliza correctamente los conceptos de programación OO y JAVA vistos en clase, tales como: encapsulación, polimorfismo, herencia, abstracción, uso de interfaces, excepciones y enumeraciones.	**4 para >2.0 pts** **Bien** La mayor parte del código hace un buen uso de los conceptos OO y de las capacidades del lenguaje JAVA, pero algunas partes menores del código deben mejorarse en estos aspectos.	**2 para >0 pts** **Por mejorar** Gran parte del código proporcionado no utiliza correctamente los conceptos OO, o muestra una falta de dominio de las posibilidades que ofrece el lenguaje JAVA.	6 pts
Modelo de clases UML KPI: 1.1	**6 para >4.0 pts** **Excelente** El modelo UML es correcto (contiene todas las clases, métodos, propiedades o relaciones para resolver el problema) y se respeta el formato UML.	**4 para >2.0 pts** **Bien** Falta informaciones menores en el modelo UML y/o hay errores menores.	**2 para >0 pts** **Por mejorar** El modelo UML es bastante incompleto o no respeta el estándar UML.	6 pts
Correspondencia entre diagrama UML y código KPI: 1.2	**3 para >2.0 pts** **Excelente** El código proporcionado y el diagrama UML proporcionado corresponden.	**2 para >1.0 pts** **Bien** Hay pequeñas incoherencias entre el modelo UML proporcionado y el código.	**1 para >0 pts** **Por mejorar** Hay muchas incoherencias entre el modelo UML y el código proporcionado.	3 pts
Documentación KPI: 1.2	**3 para >2.0 pts** **Excelente** Existe documentación en formato javadoc para todas las clases y métodos públicos.	**2 para >1.0 pts** **Bien** Faltan algunos detalles de la documentación o el cumplimiento del formato Javadoc.	**1 para >0 pts** **Por mejorar** Documentación incompleta o formato incorrecto.	3 pts
Pruebas unitarias KPI 2.1	**12 para >8.0 pts** **Excelente** Los casos de prueba cubren todos los casos de uso, clases y métodos importantes del software de forma organizada y coherente. JUnit se utiliza correctamente.	**8 para >4.0 pts** **Bien** Faltan algunas pruebas unitarias o algunas pruebas tienen problemas con el código.	**4 para >0 pts** **Por mejorar** Faltan a pruebas importantes o son incorrectas, o es imposible ejecutar las pruebas con la configuración proporcionada.	12 pts
Explicación y justificación de las opciones de diseño elegidas KPI 1.3, 2.2	**6 para >4.0 pts** **Excelente** Las opciones de diseño elegidas son pertinentes y están bien justificadas de forma concisa en el readme. El readme completa adecuadamente el modelo UML suministrado	**4 para >2.0 pts** **Bien** Algunas opciones de diseño son irrelevantes o no son realmente óptimas, o las explicaciones proporcionadas no son realmente convincentes y/o no son muy claras.	**2 para >0 pts** **Por mejorar** La mayoría de las decisiones tomadas son cuestionables y las justificaciones dadas no son convincentes.	6 pts
Trabajo en grupo KPI: 5.1	**3 para >2.0 pts** **Excelente** El trabajo está bien distribuido entre los miembros del equipo y esto se refleja tanto en Git.	**2 para >1.0 pts** **Bien** El trabajo está bastante bien distribuido en general pero hay un ligero desequilibrio entre los miembros del grupo.	**1 para >0 pts** **Por mejorar** El trabajo está desequilibrado o existe una falta de cooperación por parte de uno o más miembros del grupo. Si no hay una contribución significativa de una o varias personas, se puede aplicar una penalización de puntos en el total de la nota o poner NCR.	3 pts
Uso de GIT y GitHub KPI: 5.2	**3 para >2.0 pts** **Excelente** GIT se ha utilizado correctamente (los tamaños, las frecuencia, ramas y los mensajes de commits son relevantes).	**2 para >1.0 pts** **Bien** El uso de git es correcto en general, pero algunos mensajes de commits no son claros o las commits deberían ser un poco más frecuentes.	**1 para >0 pts** **Por mejorar** Se hace un uso moderado de GIT, pero hay problemas recurrentes con los commits en mensajes o frecuencia. Si no se usa git lo suficiente, se puede aplicar una penalización de puntos en el total de la nota o poner NCR (especialmente si se utiliza la interfaz de github en lugar de git directamente)	3 pts
				Puntos totales: 54