## Examen 111Mil - Examen y Preguntas

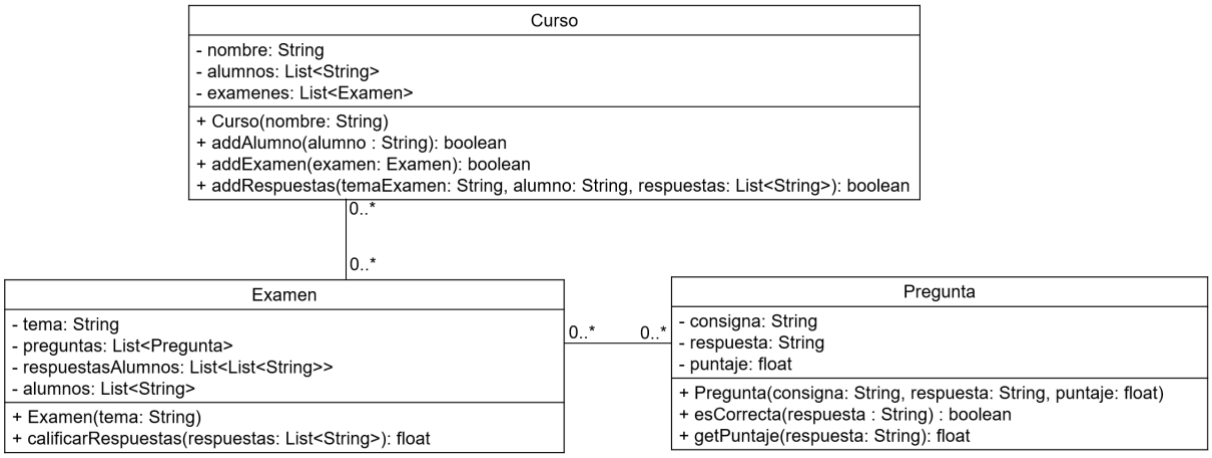
Luis es dueño del instituto de Inglés “Blue Skies”. Al principio, cuando tenía un solo curso de alumnos, Luis podía fácilmente definir las preguntas de los exámenes con los que evaluaba a sus alumnos. Sin embargo, en los últimos tiempos debido a la gran cantidad de alumnos y los cursos que debió abrir, Luis necesita una forma fácil de mantener organizados los exámenes que toma de cada curso, qué preguntas contiene cada examen, qué alumnos rindieron los exámenes y cómo le fue a los alumnos que rindieron.

Como sabe que cursamos el programa 111Mil se puso en contacto con nosotros para que lo ayudemos a construir el sistema, que en esta primera etapa del proyecto deberá registrar los exámenes, las preguntas que lo componen y las respuestas de los alumnos.

### **Ejercicio 1.** Implementar desde el diagrama de clases

Dado el siguiente diagrama UML del sistema, implementar la clase `Examen`. Para ello tener en cuenta que:

- Las listas se deben inicializar vacías dentro del constructor.
- El método `calificarRespuestas` debe sumar los puntajes obtenidos para cada una de las respuestas en la lista pasada por parámetro. Asumir que las preguntas y las respuestas se encuentran en la misma posición de las listas. Por ejemplo, la respuesta en la posición 0 se corresponde con la pregunta de la posición 0. Para esto se puede utilizar el método getPuntaje de la clase `Pregunta`, el cual dado una respuesta retorna el puntaje correspondiente.




### **Ejercicio 2.** Implementar un método a partir de un enunciado

A Luis le gustó la primera implementación del programa que le resuelve sus problemas. Pero se le ocurrió otra funcionalidad que quiere agregar al sistema:

- Dado un tema de examen y una calificación, obtener la lista de alumnos del curso que no rindieron el examen o obtuvieron una calificación menor a la dada.

Implemente la funcionalidad indicando a qué clase corresponde el/los método/s que satisface/n dicha funcionalidad.

Tenga en cuenta que la clase Examen además de los métodos implementados en el Ejercicio 1 cuenta con los siguientes métodos:

- El método `addRespuestas` de `Curso` busca el examen del tema correspondiente e invoca el método `addRespuestas` de dicho examen con el alumno y las respuestas pasadas por parámetro.
- El método `addRespuestas` de `Examen` agrega las respuestas de un alumno solo si el alumno no ha respondido ya el examen. Es decir, para cada alumno se almacena una única respuesta.
- El método `addPregunta` agrega una pregunta al `Examen`.

## **Ejercicio 3.** Análisis y seguimiento de código

Considerando el siguiente método en la clase `Curso`:

```
public List<String> XXX(String alumno){

    List<String> xx = new ArrayList<String>();
    Iterator<Examen> it = examenes.iterator();
    while(it.hasNext()){
        Examen e = it.next();
        if(e.getCalificacion(alumno) > -1)
            xx.add(e.getTema());
    }
    return xx;
}
```

¿Qué se imprimirá al ejecutar el siguiente código? Tenga en cuenta que:

- El método `addRespuestas` de `Curso` busca el examen del tema correspondiente e invoca el método `addRespuestas` de dicho examen con el alumno y las respuestas pasadas por parámetro.
- El método `addPregunta` agrega una pregunta al `Examen`.

```
Curso c = new Curso("ingles1");
c.addAlumno("pedro");
c.addAlumno("luis");

Examen e1 = new Examen("verbos irregulares");
e1.addPregunta(pregunta1); //asumir que pregunta1 ya se encuentra creada

Examen e2= new Examen("verbos modales");
```

```
e2.addPregunta(pregunta3); //asumir que pregunta3 ya se encuentra creada

c.addExamen(e1);
c.addExamen(e2);

c.addRespuestas("verbos irregulares", "pedro", respuestasPedro); //asumir que
respuestasPedro ya se encuentra creada
c.addRespuestas("verbos irregulares", "luis", respuestasLuis); //asumir que respuestasLuis
ya se encuentra creada
c.addRespuestas("verbos irregulares", "luis", respuestasLuis); //asumir que respuestasLuis
ya se encuentra creada

c.addRespuestas("verbos modales", "luis", respuestasLuisModales); //asumir que
respuestasLuisModales ya se encuentra creada

System.out.println(c.yyy("pedro"));
System.out.println(c.yyy("luis"));
```

## **Ejercicio 4.** Javadoc

Elaborar la documentación técnica utilizando Javadoc del método `calificarRespuesta` implementado en el Ejercicio 1. Incluya tanto la descripción del método como los tags que correspondan.

## **Ejercicio 5.** Interpretación de DER y SQL


Dado el diagrama de entidad-relación parcial correspondiente al sistema que estamos desarrollando, Luis quiso obtener para los alumnos que rindieron examen del tema “verbos irregulares” y respondieron más de dos preguntas su nombre y cantidad de preguntas respondidas. Los alumnos deben estar ordenados de forma descendente de acuerdo a la cantidad de preguntas respondidas. Para ello, Luis escribió la siguiente consulta:

```
SELECT alumno FROM respuesta
       INNER JOIN pregunta ON respuesta.pregunta_idPregunta = pregunta.idPregunta
                     WHERE pregunta.examen_idExamen
                     IN (SELECT idExamen FROM examen
                                   WHERE tema = "verbos irregulares")
ORDER BY alumno
```

Sin embargo, la consulta no obtiene lo que Luis quiere. Modificar la consulta para cumplir con la especificación de Luis.

```
SELECT alumno, FROM respuesta
INNER JOIN pregunta ON respuesta.pregunta_idPregunta = pregunta.idPregunta
      WHERE pregunta.Examen_idExamen
IN (SELECT idExamen FROM examen
      WHERE tema = "verbos irregulares")
```

## **Ejercicio 6.** Hibernate

Dado el diagrama de entidad-relación presentado en el ejercicio anterior y el diagrama UML presentado en el ejercicio 1, escriba la línea del archivo de mapeo de Hibernate (en formato XML o anotación) correspondiente al mapeo del atributo “consigna” de la clase y tabla Pregunta.

## Solución

### Ejercicio 1. Implementar desde el diagrama de clases

```
public class Examen{
    private String tema;
    private List<Pregunta> preguntas;
    private List<List<String>> respuestasAlumnos;
    private List<String> alumnos;

    public Examen(String tema){
        this.tema  = tema;
        preguntas = new ArrayList<>();
        respuestasAlumnos= new ArrayList<>();
        alumnos = new ArrayList<>();
    }

    public float calificarRespuestas(List<String> respuestas){
        float puntaje = 0;
        Iterator<Pregunta> itPreguntas = preguntas.iterator();
        Iterator<String> itRespuestas = respuestas.iterator();
        while(itPreguntas.hasNext()){
            Pregunta p = itPreguntas.next();
            String r = itRespuestas.next();
            puntaje += p.getPuntaje(r);
        }
        return puntaje;
    }

}
```

### **Ejercicio 2.** Implementar un método a partir de un enunciado

**Método en la clase Curso:**

```
public List<String> getAlumnosCalificacionMenorEnTema(String tema, float
calificacion){
    List<String> alumnos = new ArrayList<String>();
    Iterator<Examen> it = examenes.iterator();
    while(it.hasNext()){
        Examen e = it.next();
        if(e.esDeTema(tema)){
            Iterator<String> itAlumnos = this.alumnos.iterator();
            while(itAlumnos.hasNext()){
                String alumno = itAlumnos.next();
                if(e.getCalificacion(alumno) < calificacion)
                    alumnos.add(alumno);
            }
        }
```

```
    }
    return alumnos;
}
```

**Es una posible solución, otra solución podría ser separar la búsqueda del examen en un método aparte. No necesitan ningún método adicional de la clase Examen. No se pide que los nombres de los alumnos estén sin repetir.**

## Ejercicio 3. Análisis y seguimiento de código

```
[verbos irregulares]
[verbos irregulares,verbos modales]
```

## Ejercicio 4. Javadoc

```
/**
 *  Suma los puntajes obtenidos para cada una de las respuestas en la lista pasada por
 * parámetro. Asume que las preguntas y las respuestas se encuentran en la misma posición
 * de las listas.
 * @param  respuestas List de respuestas a calificar
 * @return puntaje obtenido
 */
```

**Puede haber variaciones en la descripción, la cual puede ser copiada del enunciado del Ejercicio 1.**

## Ejercicio 5. Interpretación de DER y SQL

```
SELECT alumno, count(*) FROM respuesta
INNER JOIN pregunta ON respuesta.pregunta_idPregunta = pregunta.idPregunta
      WHERE pregunta.Examen_idExamen
IN (SELECT idExamen FROM examen
      WHERE tema = "verbos irregulares")
      group by alumno
      having count(*) > 2
ORDER BY count(*) desc
```

**En rojo lo que deben agregar/modificar de la consulta dada.**

## Ejercicio 6. Hibernate

```
<property name=”consigna” type=”java.lang.String”/>
```