El código se encuentra en:

<https://github.com/juanga16/p111mil/tree/master/Java/p111mil.Cursos>

Examen 12/12/2018.

Sebastián es profesor de informática. Al principio, cuando tenía un solo curso de alumnos, Sebastián podía fácilmente tener registro de las unidades y los ejercicios que daba a sus alumnos. Sin embargo, en los últimos tiempos Sebastián empezó a dar clases en más cursos, de modo que ahora necesita una forma fácil de mantener organizadas las unidades temáticas que enseña en cada curso, los ejercicios que da a los alumnos para cada unidad, qué alumnos resolvieron los ejercicios y cómo les fue a los alumnos con las resoluciones.  
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 las unidades, los ejercicios que lo componen y las resoluciones de los alumnos.

Ejercicio 1 - Implementar desde el diagrama de clases

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

a. Las listas se deben inicializar vacías dentro del constructor.  
b. El método calificarEjercicios debe sumar los puntajes obtenidos para cada una de las respuestas en la lista pasada por parámetro. Asumir que los ejercicios y las respuestas se encuentran en la misma posición de las listas. Por ejemplo, la respuesta en la posición 0 se corresponde con el ejercicio de la posición 0. Para esto se puede utilizar el método getPuntaje de la clase Ejercicio, el cual dada una respuesta retorna el puntaje correspondiente.

Clase Unidad

public Unidad(String tema) {

this.tema = tema;

this.ejercicios = new ArrayList<Ejercicio>();

this.respuestasAlumnos = new ArrayList<List<String>>();

this.alumnos = new ArrayList<String>();

}

/\*\*

\* Dado un conjunto de respuestas de ejercicios retorna la puntuacion de la unidad

\* @param respuestas

\* @return puntaje de la unidad segun esas respuestas

\*/

public float calificarEjercicios(List<String> respuestas) {

float calificacion = 0;

for(int i=0; i<respuestas.size(); i++) {

String respuesta = respuestas.get(i);

Ejercicio ejercicio = this.ejercicios.get(i);

calificacion = calificacion + ejercicio.getPuntaje(respuesta);

}

return calificacion;

}

Clase Ejercicio

public boolean esCorrecto(String respuesta) {

return this.respuesta.equals(respuesta);

}

public float getPuntaje(String respuesta) {

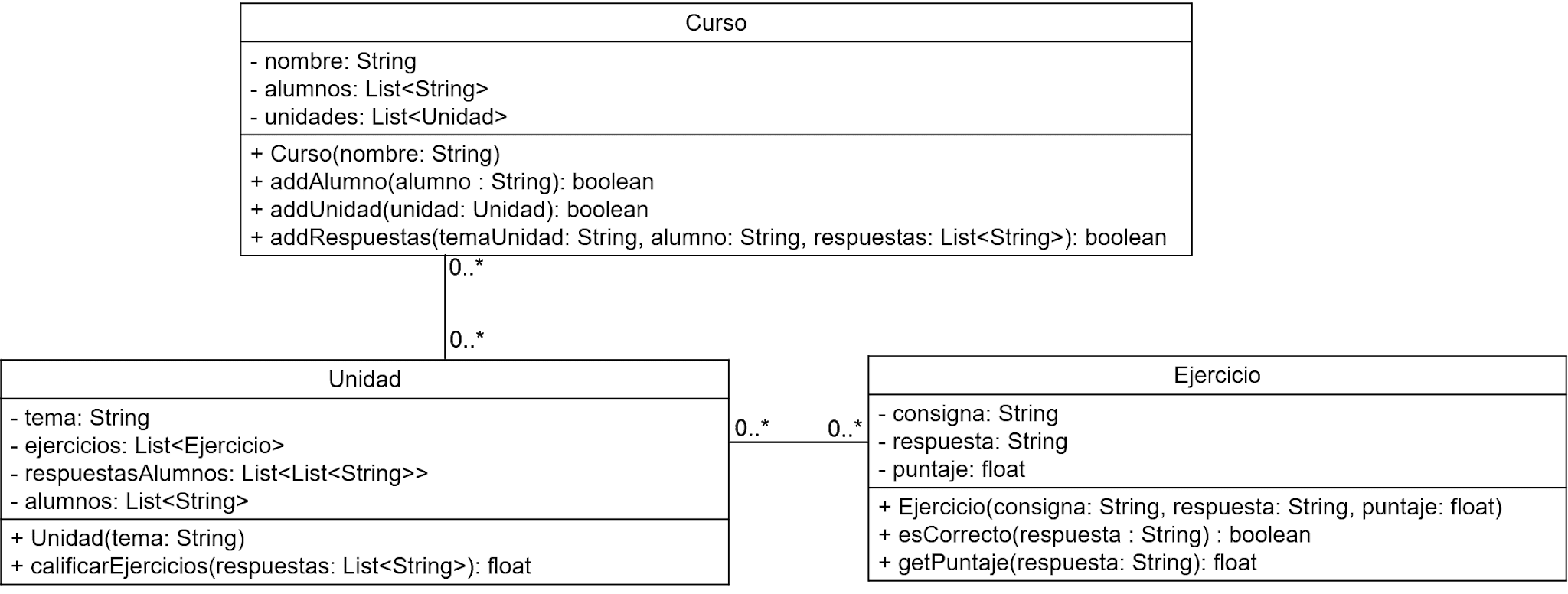
if (esCorrecto(respuesta)) {

return this.puntaje;

}

return 0;

}



Ejercicio 2 - Implementar una funcionalidad a partir de un enunciado

A Sebastián 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 unidad y una calificación, obtener la lista de alumnos del curso que no obtuvieron una calificación mayor 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 Unidad además de los métodos implementados en el Ejercicio 1 cuenta con los siguientes métodos:  
  
- El método getCalificacion de la clase Unidad retorna -1 si el alumno pasado por parámetro no resolvió los ejercicios correspondientes, o un valor mayor o igual a cero si los resolvió.  
- El método getTema de Unidad retorna el tema de la unidad.  
- El método esDeTema de Unidad retorna verdadero si el tema pasado por parámetro es igual al tema de la Unidad.

Indique en qué clase/s iría dicha funcionalidad e implemente el/los método/s necesarios para satisfacer este nuevo requerimiento

Clase Unidad

public boolean esDeTema(String tema) {

return this.tema.equals(tema);

}

public String getTema() {

return this.tema;

}

public float getCalificacion(String alumno) {

float calificacion = -1;

int posicionAlumno = this.getPosicionAlumno(alumno);

if (posicionAlumno != -1) {

calificacion = calificarEjercicios(this.respuestasAlumnos.get(posicionAlumno));

}

return calificacion;

}

private int getPosicionAlumno(String alumno) {

int posicionAlumno = -1;

for(int i = 0; i<this.alumnos.size(); i++) {

if(this.alumnos.get(i).equals(alumno)) {

posicionAlumno = i;

break;

}

}

return posicionAlumno;

}

public boolean addRespuestas(String alumno, List<String> respuestas) {

int posicionAlumno = this.getPosicionAlumno(alumno);

if (posicionAlumno == -1) {

this.alumnos.add(alumno);

this.respuestasAlumnos.add(respuestas);

return true;

}

return false;

}

void addEjercicio(Ejercicio ejercicio) {

this.ejercicios.add(ejercicio);

}

Ejercicio 3 - Seguimiento de Código

Considerando el siguiente método en la clase Curso:  
  
public List<String> xxx(String alumno){  
  
List<String> xx = new ArrayList<String>();  
Iterator<Unidad> it = unidades.iterator();  
while(it.hasNext()){  
Unidad u = it.next();  
if(u.getCalificacion(alumno) > -1)  
xx.add(u.getTema());  
}  
return xx;  
}  
  
¿Qué se imprimirá al ejecutar el siguiente código? Tenga en cuenta que:  
  
- El método getCalificacion de la clase Unidad retorna -1 si el alumno no resolvió los ejercicios de la unidad, y un valor mayor o igual a cero si los resolvió.  
- El método getTema de Unidad retorna el tema de la unidad.  
- El método addRespuestas de Curso busca la unidad del tema correspondiente e invoca el método addRespuestas de dicha unidad con el alumno y las respuestas pasadas por parámetro.  
  
Curso c = new Curso("Computación Primero");  
c.addAlumno("pedro");  
c.addAlumno("luis");  
  
Unidad u1 = new Unidad("uso de procesador de texto");  
u1.addEjercicio(ejercicio1); //asumir que ejercicio1 ya se encuentra creado  
u1.addEjercicio(ejercicio2); //asumir que ejercicio2 ya se encuentra creado  
  
Unidad u2= new Unidad("uso de planilla de cálculo");  
u2.addEjercicio(ejercicio3); //asumir que ejercicio3 ya se encuentra creado  
u2.addEjercicio(ejercicio4); //asumir que ejercicio4 ya se encuentra creado  
  
c.addUnidad(u1);  
c.addUnidad(u2);  
  
c.addRespuestas("uso de procesador de texto", "pedro", respuestasPedro); //asumir que respuestasPedro ya se encuentra creada  
c.addRespuestas("uso de planilla de cálculo", "luis", respuestasLuis); //asumir que respuestasLuis ya se encuentra creada  
c.addRespuestas("uso de planilla de cálculo", "luis", respuestasLuis); //asumir que respuestasLuis ya se encuentra creada  
  
c.addRespuestas("uso de procesador de texto", "luis", respuestasLuisTexto); //asumir que respuestasLuisTexto ya se encuentra creada  
  
System.out.println(c.xxx("luis"));  
System.out.println(c.xxx("pedro"));

¿Qué imprime?

El método xxx dado un alumno retorna una lista con todos los temas que respondió. Salida de la consola:

compile:

run:

[uso de procesador de texto, uso de planilla de cálculo]

[uso de procesador de texto]

BUILD SUCCESSFUL (total time: 0 seconds)

Ejercicio 4 - Documentación con Javadoc

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

Javadoc

/\*\*

\* Dado un conjunto de respuestas de ejercicios retorna la puntuacion de la unidad

\* @param respuestas

\* @return puntaje de la unidad segun esas respuestas

\*/

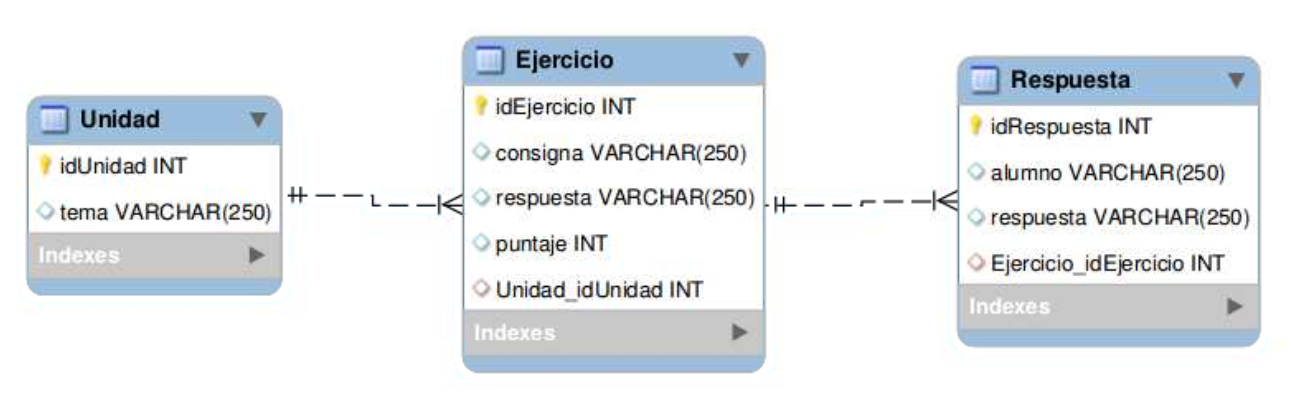
public float calificarEjercicios(List<String> respuestas) {

…

}

Ejercicio 5 - Base de Datos

Se ha modelado el siguiente diagrama de Entidad Relación.



Consulta SQL

Dado el diagrama de entidad-relación parcial correspondiente al sistema que estamos desarrollando, Sebastián quiso obtener para los alumnos que resolvieron más de dos ejercicios de la unidad del tema “uso de procesador de texto”, su nombre y cantidad de ejercicios resueltos. Los alumnos deben estar ordenados de forma descendente de acuerdo a la cantidad de ejercicios resueltos. Para ello, Sebastián escribió la siguiente consulta:  
  
SELECT alumno FROM respuesta  
INNER JOIN ejercicio ON respuesta.ejercicio\_idEjercicio = ejercicio.idEjercicio  
WHERE ejercicio.unidad\_idUnidad  
IN (SELECT idUnidad FROM unidad  
WHERE tema <> "uso de procesador de texto")  
ORDER BY alumno  
  
Sin embargo, la consulta no obtiene lo que Sebastián quiere. Modificar la consulta para cumplir con la especificación de Sebastián.

SQL

SELECT r.alumno, COUNT(r.\*)

FROM Respuesta r

INNER JOIN Ejercicio e ON e.idEjercicio = r.Ejercicio\_idEjercicio

INNER JOIN Unidad u ON e.Unidad\_idUnidad = u.idUnidad

WHERE u.tema = “Uso de procesador de texto”

GROUP BY r.alumno

HAVING COUNT(r.\*) > 2

ORDER BY COUNT(e.\*)

Ejercicio 6 - Mapeo a 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 Ejercicio.

Hibernate

<property name="consigna" type="string">

<column name=" consigna " />

</property>