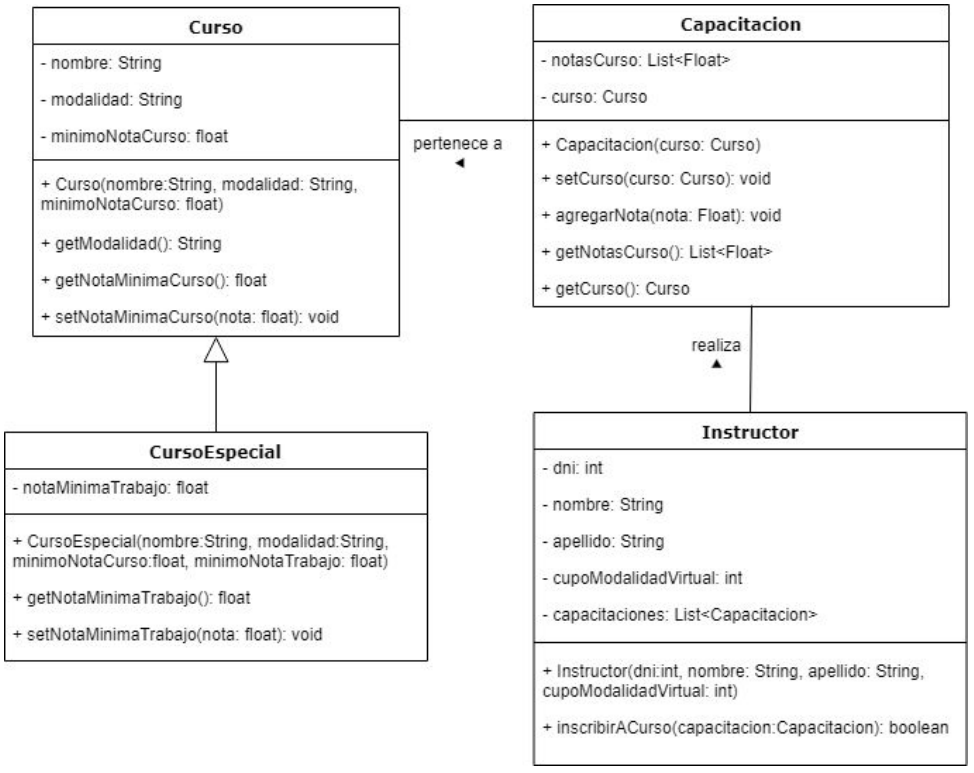
Examen 111Mil – Capacitaciones

La Universidad Luso-Argentina (ULA) está empezando a migrar el sistema de gestión de capacitaciones a instructores de programación junto con el registro de cursos en los que se encuentran inscriptos. El sistema permite registrar los instructores e inscribirlos en diferentes cursos. Cada curso tiene una modalidad de dictado que puede ser virtual o presencial, y una nota mínima requerida para la aprobación de dicho curso. Para aprobar el curso, el instructor debe rendir un único examen parcial y obtener una nota igual o superior a la nota mínima requerida. Existen también cursos *especiales*, en los cuales además hay que realizar un Trabajo Práctico Especial que puede tener al menos una entrega (si hay correcciones, habrá re-entregas sucesivas). Para aprobar un curso especial es necesario aprobar el examen parcial del curso con una nota superior o igual a la nota mínima requerida y, además, tener aprobado el Trabajo Práctico Especial (junto con sus re-entregas) con una nota igual o superior a la nota mínima requerida para aprobar dicho trabajo. Como en la ULA saben que cursamos el programa 111Mil nos invitaron a ser parte del desarrollo.

Ejercicio 1. Implementar desde el diagrama de clases

Integrantes del programa de la ULA realizaron un diagrama UML preliminar del sistema y nos pidieron que implementemos la clase *CursoEspecial* según el diagrama (el resto de las clases serán implementadas por integrantes de la ULA).



Posible Solución

Ejercicio 1. Implementar desde el diagrama de clases

```
public class CursoEspecial extends Curso{
	private float notaMinimaTrabajo;

	public CursoEspecial(String nombre, String modalidad, float
minimoNotaCursada, float minimoNotaTrabajo) {
		super(nombre, modalidad, minimoNotaCursada);
		this.notaMinimaTrabajo=minimoNotaTrabajo;

	 }


	public float getNotaMinimaTrabajo() {
		return notaMinimaTrabajo;
	}

	public void setNotaMinimaTrabajo(float notaMinimaTrabajo) {
		this.notaMinimaTrabajo = notaMinimaTrabajo;
	}
   }
```

Ejercicio 2. Implementar un método a partir de un enunciado

Programar en Java la funcionalidad para decidir, dada una lista de notas (`List<Float> notas`), si un Curso es aprobado o no. Para aprobar un curso, la nota obtenida por el instructor debe ser igual o superior que la nota mínima requerida. Para aprobar un curso especial, además de obtener una nota igual o superior que la nota mínima requerida, el promedio de las notas de entregas del Trabajo Práctico Especial debe ser igual o superior a la nota mínima requerida para aprobar el trabajo.
Ejemplo: Supongamos la siguiente lista de notas [7, 4, 8, 9]. La primera posición de la lista se corresponde con la nota del curso (7), la siguiente posición es la nota obtenida en el Trabajo Práctico Especial (4), y las siguientes posiciones se corresponden con las notas de re-entregas del Trabajo Práctico Especial (8, 9). Además, supongamos que la mínima nota para aprobar el curso es 4 y la nota mínima del Trabajo Especial es 7. La nota obtenida del curso es 7, la cual es superior a 4; y la nota promedio de las entregas del Trabajo Especial es (4+8+9)/3= 7, que es igual a 7. En este caso, el curso es aprobado por cumplir con ambas condiciones.

Implemente los métodos que considere necesarios indicando para cada uno de ellos a qué clase corresponde.

Posible Solución

Ejercicio 2. Implementar un método a partir de un enunciado

```
En la clase Curso
public float obtenerCalificacionCurso(List<Float> notas) {
	return notas.get(0);
}


public boolean apruebaCurso(List<Float> notas) {
	if (obtenerCalificacionCurso(notas)>=this.minimoNotaCurso) return true;
	return false;
```

```
}
```

En la Clase CursoEspecial

```
@Override
public boolean apruebaCurso(List<Float> notas) {
   float promedio=0;
   promedio=obtenerCalificacionCurso(notas);
   if (super.apruebaCurso(notas) && (promedio>=this.notaMinimaTrabajo))
                    return true;

   return false;
}

@Override
public float obtenerCalificacionCurso(List<Float> notas) {
   float notaTrabajoEspecial=0;
   for(int i=1; i<notas.size();i++)
      notaTrabajoEspecial+=notas.get(i);
   notaTrabajoEspecial/=notas.size()-1;
   return notaTrabajoEspecial;
}
```

Ejercicio 3. Implementar y documentar

Desde la administración de la ULA decidieron limitar la cantidad de capacitaciones en cursos con la modalidad virtual que un instructor puede realizar, de modo de asegurar que el instructor realice un mínimo de cursos en forma presencial. Para ello, nos solicitaron que programemos el método llamado *inscribirACurso* en la clase *Instructor*. El método recibe como parámetro una *capacitacion* y debe verificar que el instructor no haya superado su cupo de cursos virtuales. En caso de cumplirse esta condición, el método agrega dicha capacitación a la lista *capacitaciones* y retornar true. En caso contrario retornará false.

Implemente el método solicitado y elabore la documentación técnica utilizando Javadoc. Incluya tanto la descripción del método como los *tags* que correspondan.

Posible Solución

Ejercicio 3. Implementar y documentar

En la clase Instructor

```
/**El método agrega una nueva capacitacion en la lista de capacitaciones del Instructor solo si no se ha superado el cupo del Instructor para capacitarse en cursos con modalidad “virtual”.
   *
   * @param nuevaCapacitacion capacitacion a agregar a la lista de capacitaciones
   * @return devuelve true en caso de poder agregar la capacitacion y false en caso contrario
   */

public boolean inscribirACurso (Capacitacion nuevaCapacitacion) {
Curso c = nuevaCapacitacion.getCurso();
if (c.getModalidad().equals("Virtual"))
{
        if (getCantidadCursosModalidad("Virtual")<this.cupoModalidadVirtual)
        {
        this.capacitaciones.add(nuevaCapacitacion);
        return true;
```

```
                }
                else return false;
        } else
        {
                this.capacitaciones.add(nuevaCapacitacion);
                return true;
        }

        }

        private int getCantidadCursosModalidad(String modalidad) {
        int cantidadVirtuales=0;
        Iterator<Capacitacion> iterador = this.capacitaciones.iterator();

        while (iterador.hasNext()) {
                        Capacitacion capacitacion=iterador.next();
                        Curso curso = capacitacion.getCurso();
                        if (curso.getModalidad().equals(modalidad))
                        cantidadVirtuales+=1;
        }

        return cantidadVirtuales;
        }
```

Ejercicio 4. Seguimiento de código

Integrantes de la ULA implementaron el método *obtenerIndicador()* en las clases Curso y CursoEspecial de la siguiente manera:

```
Clase Curso

public float obtenerIndicador(List<Float> notas) {
        float nota = notas.get(0);
        if (nota>this.minimoNotaCurso) return nota;
                else return this.minimoNotaCurso;
}

Clase CursoEspecial

@Override
 public float obtenerIndicador(List<Float> notas) {
                float maximo = super.obtenerIndicador(notas);
                float intermedio=0;
                for(int i=1; i<notas.size();i++)
                {
                        intermedio=notas.get(i);
                        if (intermedio>=maximo) maximo=intermedio;
                }

                return maximo;
        }
```

Nota: Suponga que en la clase Capacitacion existe el método *getNotasCurso()*, que devuelve la lista de notas obtenidas por un instructor en un curso.

```
Clase Capacitacion

public List<Float> getNotasCurso() {
        return notasCurso;
}
```

Considerando esta implementación, ¿Qué imprimirá el programa al ejecutar el siguiente código?

```
Curso c1=new Curso ("Programación", "Presencial", 6);
Curso c2=new Curso ("Bases de Datos", "Virtual", 2);
CursoEspecial c3=new CursoEspecial ("Ética", "Presencial", 7, 8);

Instructor i1 = new Instructor (123456 , "Daenerys", "Targaryen", 4);
Capacitacion cp1 = new Capacitacion (c1);
Capacitacion cp2 = new Capacitacion (c2);
Capacitacion cp3 = new Capacitacion (c3);

i1.inscribirACurso (cp1);
i1.inscribirACurso (cp2);
i1.inscribirACurso (cp3);

cp1.agregarNota((float) 4);
cp2.agregarNota((float) 8);
cp3.agregarNota((float) 7);
cp3.agregarNota((float) 3);
cp3.agregarNota((float) 5);

System.out.println(c1.obtenerIndicador(cp1.getNotasCurso()));
System.out.println(c2.obtenerIndicador(cp2.getNotasCurso()));
System.out.println(c3.obtenerIndicador(cp3.getNotasCurso()));
```

Posible Solución

Ejercicio 4. Seguimiento de código

Respuesta:

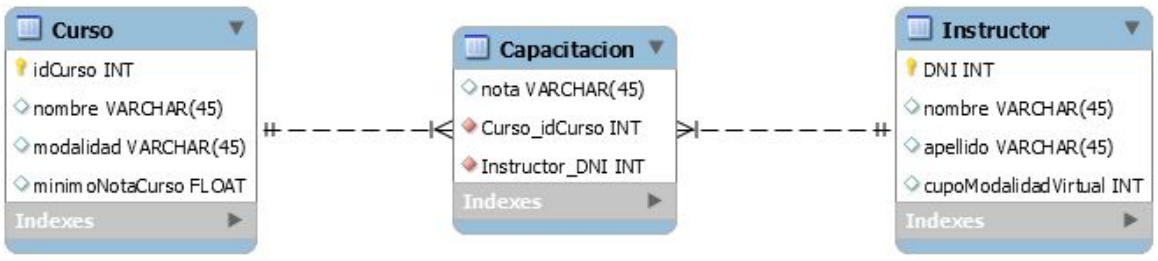
6

8

7

Ejercicio 5. Consultas SQL

Dado el diagrama de entidad-relación simplificado, escriba una consulta SQL que liste para cada instructor el promedio de notas de los cursos en los que se encuentra inscripto. Solo se deben tener en cuenta aquellos cursos con modalidad presencial. Los resultados deben estar ordenados descendentemente por el DNI de los instructores.



Además, dadas las siguientes tuplas de ejemplo, determinar el resultado de la consulta.

Curso			
1	Programación	Presencial	7
2	Bases de datos	Presencial	4
3	Ética	Virtual	8

Capacitacion		
8	1	123456
7	1	123457
5	1	123458
3	2	123456
4	2	123457
6	2	123458
9	3	123456
10	3	123457
6	3	123458

Instructor			
123456	Daenerys	Targaryen	2
123457	Jon	Snow	2
123458	Cersei	Lannister	1

Posible Solución

Ejercicio 5. Consultas SQL

```
SELECT I.DNI, AVG(CP.nota) AS Promedio FROM
        Instructor I JOIN Capacitacion CP ON I.DNI = CP.Instructor_DNI
        JOIN Curso C ON C.idCurso=CP.Curso_IdCurso
        WHERE C.modalidad="Presencial"
        GROUP BY (I.DNI)
        ORDER BY (I.DNI) DESC;
```

Resultado de la Consulta:

DNI	Promedio
123458	5.5
123457	5.5
123456	5.5