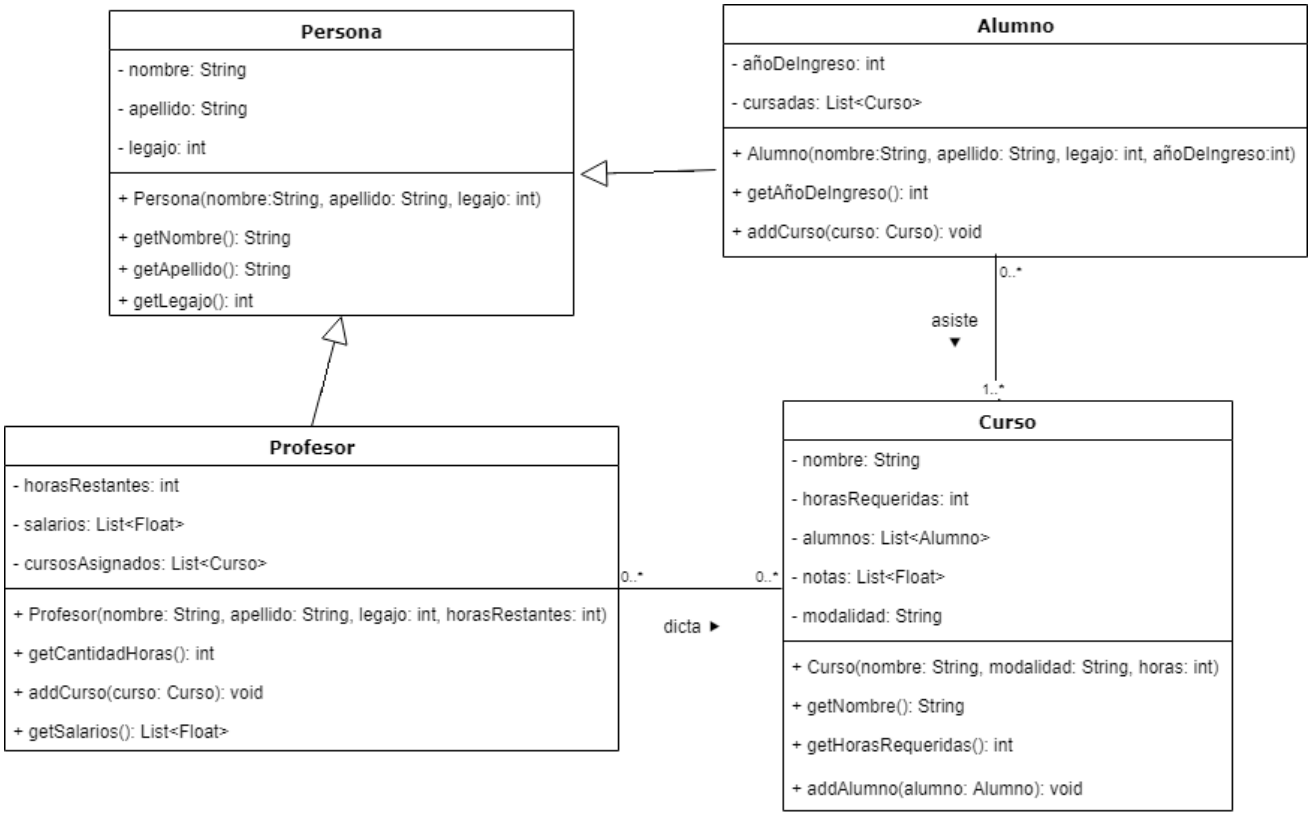
Examen 111Mil – Cursos

El Instituto de Formación Profesional San Blas (IFPSB) tiene como objetivo desarrollar un sistema para administrar los cursos que se dictan cada año. La aplicación permitirá registrar los cursos que se ofrecen, junto a los alumnos inscriptos en los mismos y los profesores que los dictan. Como saben que cursamos el programa 111Mil nos invitaron a ser parte del desarrollo.

Ejercicio 1. Implementar desde el diagrama de clases

Profesores del IFPSB realizaron un diagrama UML preliminar del sistema y nos pidieron que implementemos la clase *Curso* según el diagrama (el resto de las clases serán implementadas por el IFPSB). Los profesores nos indicaron que, una vez que hayamos implementado la clase *Curso*, ejecutarán el siguiente código para comprobar que no existen errores:

```
Curso curso=new Curso("Inteligencia Artificial", "Presencial", 40);
Alumno alumno = new Alumno("Juan", "Pérez", 20546, 2019);
curso.addAlumno(alumno);
```



Posible Solución

```
public class Curso {
   private String nombre;
   private int horasRequeridas;
   private String modalidad;
   private List<Alumno> alumnos;
   private List<Float> notas;

   public Curso(String nombre, String modalidad, int horas) {
      this.nombre=nombre;
      this.modalidad=modalidad;
```

```
        this.horasRequeridas=horas;
        this.alumnos=new ArrayList<>();
        this.notas=new ArrayList<>();
    }

    public String getNombre() {
        return this.nombre;
    }

    public int getHorasRequeridas() {
        return this.horasRequeridas;
    }

    public void addAlumno(Alumno alumno){
        this.alumnos.add(alumno);
    }
}
```

Ejercicio 2. Implementar un método a partir de un enunciado

Programar en Java la funcionalidad para obtener el promedio de horas de todos los cursos de una determinada modalidad (ej. virtual, presencial, etc.) realizados por un determinado alumno. Puede suponer que el alumno realizó al menos un curso de la modalidad buscada.

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

Posible Solución

En la clase Alumno

```
    public float obtenerPromedioHoras(String modalidad){
        Iterator<Curso> itCursos=this.cursadas.iterator();
        float totalHorasDeCursada=0.0f;
        int cursosAsistidos=0;
        while (itCursos.hasNext()) {
            Curso curso =  itCursos.next();
            if(curso.esDeModalidad(modalidad)){
                totalHorasDeCursada= totalHorasDeCursada+ curso.getHorasRequeridas();
                cursosAsistidos ++;
            }
        }
        return totalHorasDeCursada/ cursosAsistidos;
    }
```

En la clase Curso

```
public boolean esDeModalidad(String modalidad) {
      if (this.modalidad.equals(modalidad))
        return true;
return false;
    }
```

Ejercicio 3. Implementar y documentar

Desde la administración del IFPSB decidieron limitar la cantidad de cursos que cada profesor puede

dictar de modo de asegurar la calidad en el dictado de sus cursos. Para ello, nos solicitaron que modifiquemos el método llamado *addCurso* en la clase *Profesor*. El método solo debe agregar el curso a la lista *cursosAsignados* si aún el profesor cuenta con horas disponibles por semana. Tenga en cuenta que cada curso contiene las horas semanales requeridas. En caso de cumplirse esta condición, el método deberá agregar el curso en la lista *cursosAsignados*, decrementar las horas disponibles del profesor 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

```
 /**El método agrega un nuevo curso en la lista de cursos asignados al Profesor
solo si aún no se superó el límite máximo de horas disponibles del profesor.
     *
     * @param nuevoCurso curso a agregar a la lista de cursos
     * @return devuelve true en caso de poder agregar el curso y false en caso contrario
     */
    public boolean addCurso (Curso nuevoCurso){
      int horasCurso = nuevoCurso.getHorasRequeridas();
       if((this.horasRestantes - horasCurso) >=0){
          this.horasRestantes= this.horasRestantes - horasCurso;
          this.cursosAsignados.add(nuevoCurso);
          return true;
       }
       return false;
    }
```

Ejercicio 4. Seguimiento de código

Los docentes del IFPSB sobrescribieron el método toString en las clases Persona y Profesor de la siguiente manera:

Clase Persona

```
 @Override
   public String toString() {
          return this.nombre+" "+ this.apellido+" "+ this.legajo;
  }
```

Clase Profesor

```
  @Override
    public String toString() {
       return super.toString()+" "+ this.horasRestantes+" "+ this.cursosAsignados.size();
   }
```

Considerando estos cambios, ¿Qué imprimirá el programa al ejecutar el siguiente código? Recuerde que el método *addCurso* asigna un curso al Profesor siempre y cuando no se haya alcanzado el número máximo horas disponibles para ese profesor y decrementa las horas disponibles del profesor.

```
  Curso c1=new Curso(“Programación”, “Presencial”, 4);
  Curso c2=new Curso(“Bases de Datos”, “Virtual”, 2);
  Curso c3=new Curso(“Técnicas", “Presencial”, 6);

  Profesor p1=new Profesor ("Daenerys", "Targaryen", 123456, 40);
  Profesor p2=new Profesor ("Jon", "Snow", 123457, 30);
  Profesor p3=new Profesor ("Cersei", "Lannister", 123458, 20);
```

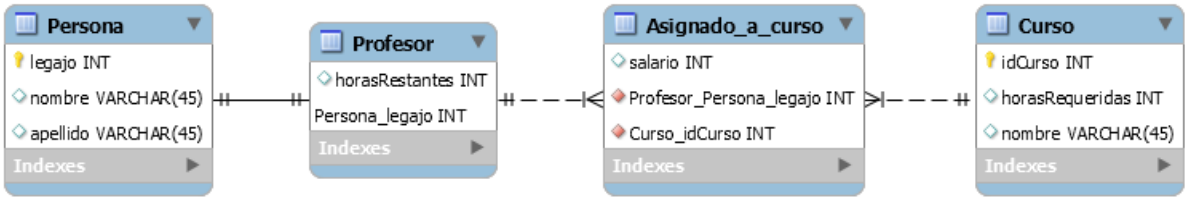
```
p1.addCurso(c1);
p2.addCurso (c2);
p3.addCurso (c3);

System.out.println(p1);
System.out.println(p2);
System.out.println(p3);
```

Posible Solución

Daenerys Targaryen 123456 36 1
Jon Snow 123457 28 1
Cersei Lannister 123458 14 1

Ejercicio 5. Consultas SQL

Dado el diagrama de entidad-relación parcial, escriba una consulta SQL que liste para cada profesor el promedio de los salarios de los cursos que dicta. Solo se deben tener en cuenta aquellos cursos que requieren 20 horas o menos. Los resultados deben estar ordenados ascendentemente por el legajo de los profesores. Además, dadas las siguientes tuplas de ejemplo, determinar el resultado de la consulta.



Persona		
123456	Daenerys	Targaryen
123457	Jon	Snow
123458	Cersei	Lannister

Profesor	
50	123456
65	123457
75	123458

Asignado_a_curso		
12.000	123456	1
10.000	123457	1
16.000	123458	1
9.000	123456	2
12.000	123457	2
12.000	123458	2
10.000	123456	3
8.000	123457	3
2.000	123458	3

Curso		
1	10	Programacion
2	25	Bases de datos
3	15	Tecnicas

Posible Solución

```
SELECT p.Persona_legajo, AVG(a.salario) FROM Profesor p
INNER JOIN Asignado_a_curso a ON (p.Persona_legajo=a.Profesor_Persona_legajo)
INNER JOIN Curso c ON (a.Curso_idCurso=c.idCurso) WHERE c.horasRequeridas <= 20
GROUP BY p.Persona_legajo
ORDER BY p.Persona_legajo;
```

Salida

123456	11.000
123457	9.000
123458	9.000

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 “horasRestantes”.

Posible Solución

```
<property name=”horasRestantes” type=”java.lang.Integer”/>
```