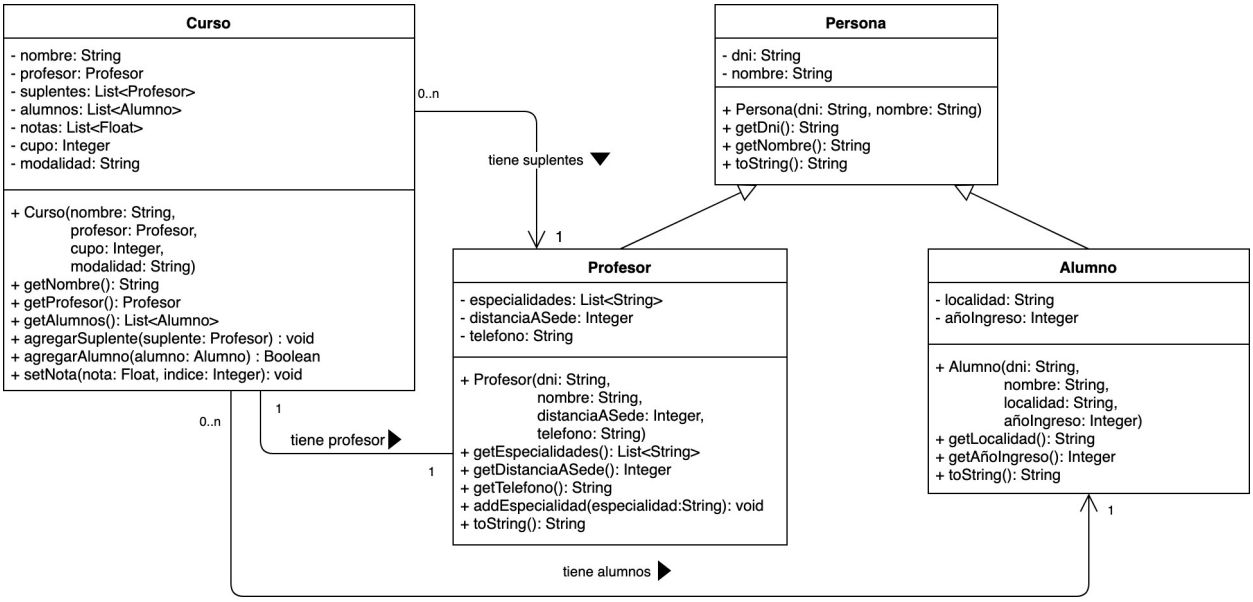
## Cursos de Oficios

La Universidad Nacional de Misiones (UNAM) brinda, al igual que muchas otras universidades en el país, cursos de oficios abiertos a la comunidad. Sin embargo, no existe hoy en día un sistema interno para administrarlos. Por ello, a los alumnos de 111Mil se les ocurrió crear una aplicación Java y donarla a la UNAM.

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

En la primer etapa de desarrollo, se creó un diagrama UML con la estructura básica de las clases de la aplicación. Se le solicita a Ud. que implemente la clase ***Profesor*** a partir del diagrama. Tome en cuenta que el método *toString* retorna un *String* construido a partir de los siguientes campos: dni, nombre, teléfono, número de especialidades y distancia a la sede, separados por coma.

```
Profesor profesor = new Profesor("1234", "Profesor Prueba", 50, "+54 223
123456789");
profesor.addEspecialidad("teatro");
System.out.println(profesor); // Debería imprimir: 1234, Profesor Prueba, +54 223
123456789, 1, 50
```




### Posible Solución

```
public class Profesor extends Persona {
    private List<String> especialidades;
    private Integer distanciaASede;
    private String telefono;

    public Profesor(String dni,
                    String nombre,
                    Integer distanciaASede,
                    String telefono) {
```

```
        super(dni, nombre);
        this.distanciaASede = distanciaASede;
        this.telefono = telefono;
        this.especialidades = new ArrayList<String>();
    }

    public String getTelefono() {
        return this.telefono;
    }

      public ArrayList<String> getEspecialidades() {
        return this.especialidades;
    }

    public Integer getDistanciaASede(){
        return this.distanciaASede;
    }

    public void addEspecialidad(String especialidad){
        return this.especialidades.add(especialidad);
    }

    public String toString() {
        return super.getDni() + ", "+
               super.getNombre() + ", "+
               this.telefono + ", "+
               this.especialidades.size() + ", "+
               this.distanciaASede;
    }
}
```

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

En invierno es normal que un profesor anuncie, horas antes de comenzar el curso, que no podrá asistir por enfermedad. Se le pide a Ud. implementar un método llamado ***getMejorSuplente*** que, utilizando la lista de profesores suplentes, obtenga aquel ***Profesor*** que se encuentre a la menor distancia de la sede. Solo deben tomarse en cuenta Profesores cuyos teléfonos no sean un String vacío. En caso de que 2 profesores con teléfono estén a la misma distancia, elegir aquel que tenga más especialidades.

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

**Posible Solución**

***En la clase Curso***

```
      public Profesor getMejorSuplente(){
            Profesor mejor = null;
            Iterator<Profesor> it=this.suplentes.iterator();
```

```
            while (it.hasNext()) {
             Profesor suplente = it.next();
             if (suplente.esMejor(mejor))
                  mejor = suplente;
             }
             return mejor;
      }
```

***En clase Profesor***

```
   public boolean esMejor(Profesor otroProfesor) {
       if (!this.getTelefono().equals(""))
        if (otroProfesor == null)
           return true;
          else
            if (this.getDistanciaASede() < otroProfesor.getDistanciaASede())
               return true;
             else
              if (this.getDistanciaASede() == otroProfesor.getDistanciaASede()
                    && this.getEspecialidades().size() >
                  otroProfesor.getEspecialidades().size())
              return true;
      return false;
   }
```

### **Ejercicio 3.** Implementar y documentar

Implemente el método ***agregarAlumno*** de la clase ***Curso***, el cual recibe un ***Alumno*** por parámetro y agrega el alumno a la lista de alumnos del curso.
Dicho método verifica que, si la modalidad del curso es **"presencial"**, no se supere el cupo establecido y que la localidad del Alumno sea **"Posadas"**. En caso de que la modalidad del curso sea **"virtual"**, no hay cupo máximo ni verificación de localidad.
El método retorna **true** si se pudo agregar al alumno y **false** en caso contrario.

Implemente el método solicitado y elabore la documentación técnica utilizando Javadoc. Incluya tanto la descripción del método así como también las anotaciones que correspondan.

**Posible Solución**

```
 /** El metodo agrega un nuevo alumno en el curso solo si el curso es
virtual, o
     * si es presencial y aun no se supero el limite maximo de inscriptos y
el
     *  alumno pertenece a la localidad de Posadas.
     *
     * @param nuevoAlumno alumno a agregar al curso
```

```
     * @return devuelve true en caso de poder agregar al alumno y false en
caso contrario
     */
    public boolean agregarAlumno(Alumno nuevoAlumno){
        if (this.modalidad.equals("presencial")) {
            if(this.alumnos.size() == this.cupo
               || !nuevoAlumno.getLocalidad().equals("Posadas")){
                return false;
            }
        }
        this.alumnos.add(nuevoAlumno);
        return true;
    }
```

### **Ejercicio 4.** Seguimiento de código

¿Qué imprimirá el programa al ejecutar el siguiente código? Recuerde que el método ***agregarAlumno*** verifica la modalidad del curso y el cupo del mismo (**Ejercicio 3**).

```
Profesor profesor = new Profesor("30.000.001", "profesor", 100, "+54 (0375)
123456789");

Curso c1 = new Curso("Programación Java", profesor, 2, "virtual");
Curso c2 = new Curso("Electricidad", profesor, 3, "presencial");
Curso c3 = new Curso("Carpintería", profesor, 1, "presencial");

Alumno a1 = new Alumno("12.345.671", "Juan", "Posadas", 2016);
Alumno a2 = new Alumno("12.345.672", "Mirta", "El Dorado",2017);
Alumno a3 = new Alumno("12.345.673", "María", "Posadas", 2018);

c1.agregarAlumno(a1);
c1.agregarAlumno(a2);
c1.agregarAlumno(a3);

c2.agregarAlumno(a1);
c2.agregarAlumno(a2);
c2.agregarAlumno(a3);

c3.agregarAlumno(a1);
c3.agregarAlumno(a2);
c3.agregarAlumno(a3);

System.out.println(c1.getAlumnos().size());
System.out.println(c2.getAlumnos().size());
System.out.println(c3.getAlumnos().size());
```

**Posible Solución**

El código imprime primero 3 (el curso c1 es virtual), luego 2, el curso c2 es presencial y no agrega a la

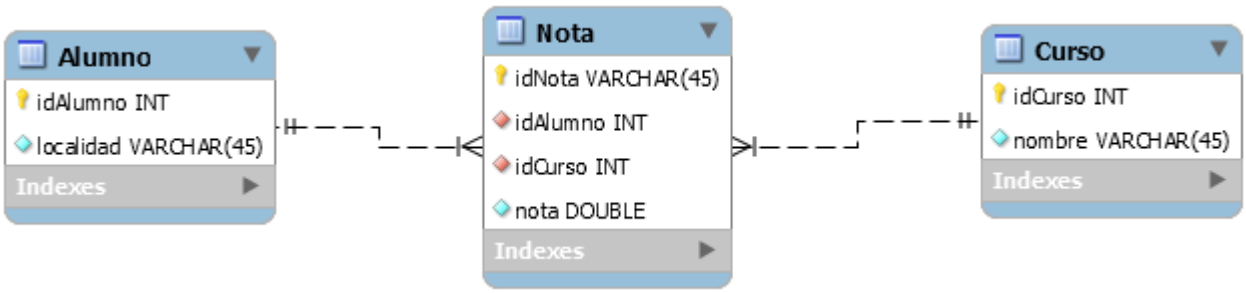
persona que no es de Posadas, y luego 1 porque el cupo de c3 es 1.

3
2
1

### **Ejercicio 5.** Consulta SQL

El departamento de estadísticas de la UNAM desea realizar un relevamiento del desempeño del alumnado en el curso de Java en el interior de la provincia. Dado el diagrama de entidades y relaciones parcial del sistema, escribir una consulta que liste para cada localidad el promedio de notas de los alumnos de dicha localidad en el curso con nombre “Java”. Como se trata del interior de la provincia, ignorar la localidad “Posadas”. Además, como se desea dar un premio a las mejores localidades, solo considerar los promedios de nota mayor o igual a 7.

Indique el resultado de aplicar su consulta a los siguientes datos.




| Alumno | |
|---|---|
| **idAlumno** | **localidad** |
| **1** | **Posadas** |
| **2** | **Oberá** |
| **3** | **Oberá** |

| Nota | | | |
|---|---|---|---|
| **idNota** | **idAlumno** | **idCurso** | **nota** |
| **1** | **1** | **1** | **8** |
| **2** | **2** | **1** | **10** |
| **3** | **3** | **1** | **4** |

| Curso |
|---|

| idCurso | nombre |
|---|---|
| 1 | Java |
| 2 | Carpintería |
| 3 | Construcción en Seco |

**Posible Solución**

**Alternativa 1:**

```
SELECT alumno.localidad, AVG(nota.nota) AS promedio FROM Alumno alumno
JOIN Nota nota ON nota.idAlumno = alumno.idAlumno
JOIN Curso curso ON nota.idCurso = curso.idCurso
WHERE alumno.localidad <> "Posadas" and curso.nombre = "Java"
GROUP BY alumno.localidad
HAVING promedio >= 7
```

**Alternativa 2:**

```
SELECT alumno.localidad, SUM(nota.nota) / COUNT(*) AS promedio
FROM Alumno alumno
JOIN Nota nota ON nota.idAlumno = alumno.idAlumno
JOIN Curso curso ON nota.idCurso = curso.idCurso
WHERE alumno.localidad <> "Posadas" and curso.nombre = "Java"
GROUP BY alumno.localidad
HAVING promedio >= 7
```

Con los datos dados, la consulta retorna:

**Oberá 7**

### 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 “localidad”.

## Posible Solución

```
<class name = "Alumno" table = "Alumno">
  <property name = "localidad" column = "localidad" type = "string"/>
</class>
```