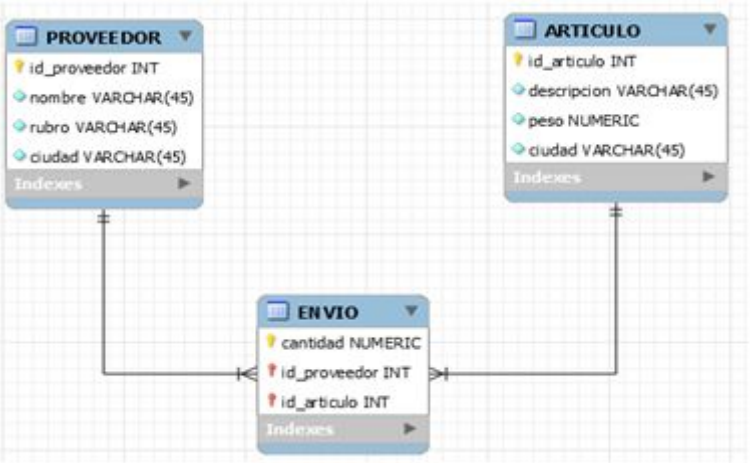
Ejercicios de Repaso para la Certificación

Ejercicios de Bases de Datos

1. **Dado el diagrama de entidad-relación, corrija la siguiente consulta SQL (ya que contiene errores) para que por cada artículo enviado, la consulta liste el nombre del proveedor junto al id del artículo.**

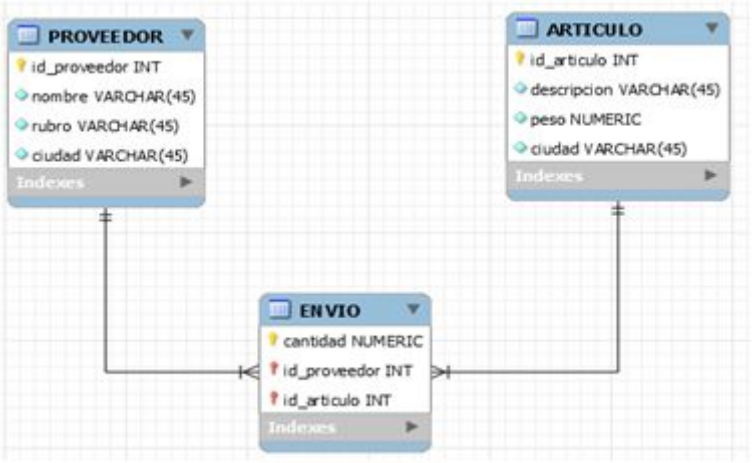


SELECT p.nombre, e.id_articulo
FROM Proveedor p JOIN Envio e ON p.nombre = e.id_articulo

Solución:
SELECT p.nombre, e.id_articulo
FROM Proveedor p JOIN Envio e ON **p.id_proveedor = e.id_proveedor**

2. **Dado el diagrama de entidad-relación, corrija la siguiente consulta SQL (ya que contiene errores) para que retorne el número total de cada artículo enviado por cada proveedor junto con el nombre del proveedor y el id del artículo.**

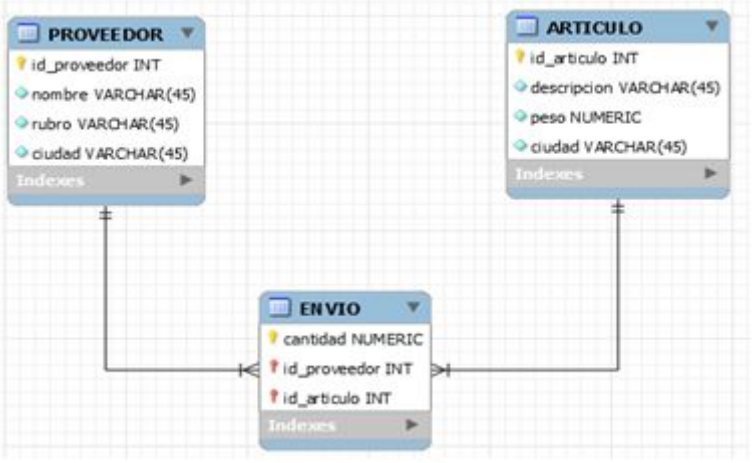
Se responderán solamente consultas técnicas sobre los ejercicios. Mandar mail a certificacion111mil@produccion.gob.ar con el asunto: "Consulta Ejercicios Certificación D2018"



SELECT SUM(e.cantidad), p.nombre, e.id_articulo
FROM Proveedor p JOIN Envio e ON p.id_proveedor = e.id_proveedor
AND e.id_articulo IN (SELECT a.descripcion FROM Articulo a)

Solución:
SELECT SUM(e.cantidad), p.nombre, e.id_articulo
FROM Proveedor p JOIN Envio e ON p.id_proveedor = e.id_proveedor
AND e.id_articulo IN (SELECT **a.id_articulo** FROM Articulo a) **GROUP BY e.id_proveedor, e.id_articulo**

3. **Dado el diagrama de entidad-relación, la consulta SQL y las siguientes tuplas de ejemplo, determinar el resultado de la consulta:**



SELECT SUM(e.cantidad), p.nombre, e.id_articulo
FROM Proveedor p JOIN Envio e ON p.id_proveedor = e.id_proveedor
AND e.id_articulo IN (SELECT a.id_articulo FROM Articulo a)
GROUP BY e.id_proveedor, e.id_articulo HAVING SUM(e.cantidad > 100)
ORDER BY p.ciudad DESC

Tuplas de ejemplo:

ARTICULO:
1,”articulo A”, 30, “Tandil”
2,”articulo B”, 50, “Balcarce”
3,”articulo C”, 10, “Olavarría”
4,”articulo D”, 60, “Pinamar”

PROVEEDOR:
1, “proveedor X”, “limpieza”, “Tandil”
2, “proveedor Y”, “higiene”, “Azul”
3, “proveedor Z”, “farmacia”, “Bolivar”
4, “proveedor W”, “limpieza”, “Mar del Plata”

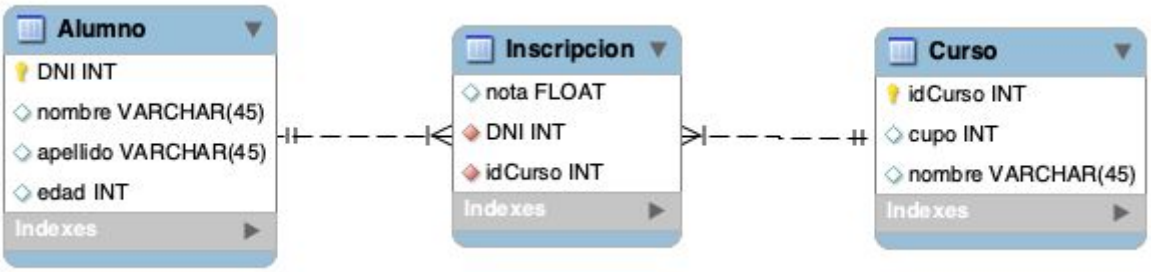
ENVIO:
120,1,1
160,1,2
90,3,1
100,4,3
80,1,1

Solución:

200,'producto X',1

160,'producto X',2

4. **El equipo administrativo que trabaja con el director de un colegio se encuentra en este momento contabilizando los estudiantes que al menos aprobaron 1 curso de los que ofrece el Programa 111Mil en el Colegio Secundario Nº1. En su base de datos existe información de los alumnos, de los cursos y de las inscripciones a dichos cursos. Dada la vista parcial del DER que se muestra a continuación, se requiere listar el nombre y apellido de cada alumno, con su DNI, edad y el nombre del o los curso/s con nota 7 ó superior. El listado debe estar ordenado alfabéticamente por el apellido del alumno. El equipo administrativo necesita que escribas la consulta SQL correspondiente.**

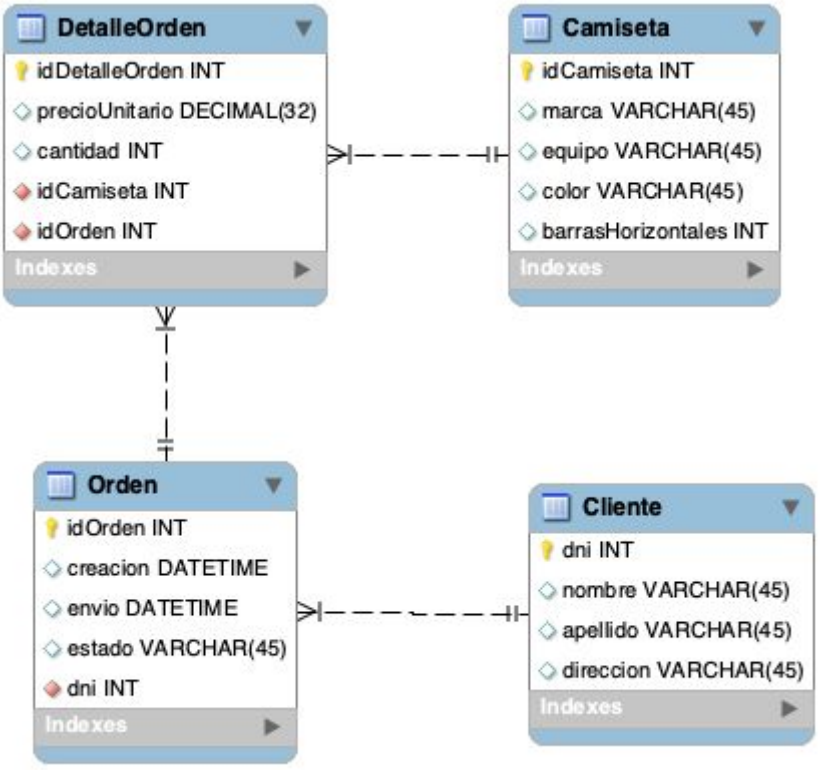


Se responderán solamente consultas técnicas sobre los ejercicios. Mandar mail a certificacion111mil@produccion.gob.ar con el asunto: "Consulta Ejercicios Certificación D2018”

Solución:

SELECT a.nombre, a.apellido, a.DNI, a.edad, c.nombre FROM Alumno a INNER JOIN Inscripcion i ON (a.DNI=i.DNI) INNER JOIN Curso c ON (i.idCurso=c.idCurso) where (i.nota>=7) ORDER BY a.apellido

5. **Dado el siguiente diagrama de entidad-relación, escriba una consulta SQL que liste los números de orden (id) de todas las órdenes pertenecientes a clientes con apellido Rodriguez.**



Solución:

SELECT o.idOrden from Orden o INNER JOIN Cliente c ON (o.dni=c.dni) where c.apellido='Rodriguez'

6. **Dada la siguiente porción de clase en Java y la sentencia de DDL de creación de la tabla escriba la línea del archivo de mapeo de Hibernate HBM en formato XML correspondiente al mapeo del atributo "monto".**

Infraccion.java

```
package poo.infracciones;
import poo.infracciones.modelos.InfraccionNomenclada;

public class Infraccion{

    private Integer id;
    private InfraccionNomenclada infraccionNomenclada;
    private Integer cantidadPuntosDescontados;
    private BigDecimal monto;
    private String observacion;
```

Script SQL de creación de la tabla

```
CREATE TABLE `Infraccion` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `cantidadPuntosDescontados` tinyint(3) unsigned DEFAULT NULL,
  `monto` decimal(8,2) unsigned NOT NULL,
  `observacion` varchar(255) DEFAULT NULL,
  `numeroActaConstatacion` int(10) unsigned NOT NULL,
  `codigoInfraccionNomenclada` int(10) unsigned NOT NULL
  PRIMARY KEY (`id`),
  KEY `fk_Infraccion_ActaConstatacion1_idx` (`numeroActaConstatacion`),
  KEY `fk_Infraccion_InfraccionNomenclada1_idx` (`codigoInfraccionNomenclada`),
  CONSTRAINT `fk_Infraccion_ActaConstatacion1`
    FOREIGN KEY (`numeroActaConstatacion`) REFERENCES `ActaConstatacion` (`numero`)
    ON DELETE NO ACTION ON UPDATE NO ACTION,
  CONSTRAINT `fk_Infraccion_InfraccionNomenclada1`
        FOREIGN KEY (`codigoInfraccionNomenclada`) REFERENCES `InfraccionNomenclada`
(`codigo`)
    ON DELETE NO ACTION ON UPDATE NO ACTION
)
```

Solución:

```
<property name="monto" type="java.math.BigDecimal" />
```

7. **Dada la siguiente porción de clase en Java y la sentencia de DDL de creación de la tabla escriba la línea del archivo de mapeo de Hibernate HBM en formato XML correspondiente al mapeo del atributo "infraccionNomenclada".**

Infraccion.java

```
package poo.infracciones;
import poo.infracciones.modelos.InfraccionNomenclada;

public class Infraccion {

    private Integer id;
    private InfraccionNomenclada infraccionNomenclada;
    private Integer cantidadPuntosDescontados;
    private BigDecimal monto;
    private String observacion;
```

Script SQL de creación de la tabla

```
CREATE TABLE `Infraccion` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `cantidadPuntosDescontados` tinyint(3) unsigned DEFAULT NULL,
  `monto` decimal(8,2) unsigned NOT NULL,
  `observacion` varchar(255) DEFAULT NULL,
  `numeroActaConstatacion` int(10) unsigned NOT NULL,
  `codigoInfraccionNomenclada` int(10) unsigned NOT NULL
  PRIMARY KEY (`id`),
  KEY `fk_Infraccion_ActaConstatacion1_idx` (`numeroActaConstatacion`),
  KEY `fk_Infraccion_InfraccionNomenclada1_idx` (`codigoInfraccionNomenclada`),
  CONSTRAINT `fk_Infraccion_ActaConstatacion1`
    FOREIGN KEY (`numeroActaConstatacion`) REFERENCES `ActaConstatacion` (`numero`)
    ON DELETE NO ACTION ON UPDATE NO ACTION,
  CONSTRAINT `fk_Infraccion_InfraccionNomenclada1`
        FOREIGN KEY (`codigoInfraccionNomenclada`) REFERENCES `InfraccionNomenclada`
(`codigo`)
    ON DELETE NO ACTION ON UPDATE NO ACTION
)
```

Solución:

```
<many-to-one name="infraccionNomenclada"
class="poo.infracciones.modelos.InfraccionNomenclada"
column="codigoInfraccionNomenclada" not-null="true" />
```

Ejercicios de Java

1. **Considere el siguiente método**

```
public String ifElseMisterioso(int x, int y){
   int z = 4;
   if (z <= x){
       z = x + 1;
   } else{
       z = z + 9;
   }
   if (z <= y){
       y++;
   }
   return z + " " + y;
}
```

Para cada una de las siguientes invocaciones, determinar el retorno de la invocación:

`ifElseMisterioso(3,20);`	
`ifElseMisterioso(4,5);`	
`ifElseMisterioso(5,5);`	
`ifElseMisterioso(6,10);`	

Solución:

`ifElseMisterioso(3,20);`	**13 21**
`ifElseMisterioso(4,5);`	**5 6**
`ifElseMisterioso(5,5);`	**13 5**
`ifElseMisterioso(6,10);`	**7 11**

2. **Considere el siguiente método:**

```
public void misterio(int[] a, int[] b){
   for (int i = 0; i < a.length; i++){
       a[i] += b[b.length - 1 - i];
   }
}
```

Dados los siguientes arreglos:

```
int[] a1 = {1, 3, 5, 7, 9};
int[] a2 = {1, 4, 9, 16, 25};
```

Determine los valores de los elementos en el arreglo a1 luego de ejecutar la siguiente invocación al método: misterio(a1,a2)

Solución:

```
a1 = {26, 19, 14, 11, 10};
a2 = {1, 4, 9, 16, 25};
```

3. **Determinar los valores almacenados en el arreglo array luego de que se ejecute el siguiente fragmento de código**

```
int [] array = {2,18,6,-4,5,1};
for(int i=0;i<array.length;i++)
    array[i] = array[i] + (array[i] / array[0]);
```

Solución:

```
array = {3,24,8,-5,6,1};
```

4. **El constructor de la clase Punto tiene dos problemas. ¿Cuáles son? Encontrar y arreglar los problemas.**

```
public class Punto{
    int x;
    int y;

    public void Punto(int xInicial, int yInicial){
        int x = xInicial;
        int y = yInicial;
    }
```

Se responderán solamente consultas técnicas sobre los ejercicios. Mandar mail a certificacion111mil@produccion.gob.ar con el asunto: "Consulta Ejercicios Certificación D2018"

```
}
```

Solución:

```
public class Punto{
    int x;
    int y;

    public Punto(int xInicial, int yInicial){
        x = xInicial;
        y = yInicial;
    }

}
```

5. **Considere la siguiente clase. Determine la salida que se produce luego de ejecutar el main.**

```
public class Raro{
   public static void main(String[] args){
       Raro raro=new Raro();
       raro.primero();
       raro.tercero();
       raro.segundo();
       raro.tercero();
   }

   public void primero(){
       System.out.println("Dentro del método primero.");
   }

   public void segundo(){
       System.out.println("Dentro del método segundo.");
       primero();
   }

   public void tercero() {
       System.out.println("Dentro del método tercero.");
       primero();
       segundo();
   }
}
```

Se responderán solamente consultas técnicas sobre los ejercicios. Mandar mail a certificacion111mil@produccion.gob.ar con el asunto: "Consulta Ejercicios Certificación D2018"

Solución:

Dentro del método primero.
Dentro del método tercero.
Dentro del método primero.
Dentro del método segundo.
Dentro del método primero.
Dentro del método segundo.
Dentro del método primero.
Dentro del método tercero.
Dentro del método primero.
Dentro del método segundo.
Dentro del método primero.

6. Considere el siguiente programa:

```
1.  public class Ejemplo{

2.      public static void mostrarReglas(){
3.          System.out.println("La primera regla ");
4.          System.out.println("del club de Java es");
5.          System.out.println("");
6.          System.out.println("no se habla del club de Java!");
7.      }

8.      public static void main(String[] args){
9.          System.out.println("Las reglas del club de Java.");
10.         Ejemplo.mostrarReglas();
11.         Ejemplo.mostrarReglas();
12.     }
13. }
```

¿Qué sucedería si se realizan los siguientes cambios a la clase Ejemplo? Considerar cada cambio de forma independiente de los otros.

Para cada cambio, considerar sólo tres posibilidades:

- "Nada": Si no ocurrirá ningún cambio en el programa.
- "Error". Si produjera que el programa no compile o que de un error durante su ejecución.
- "Salida". Si cambiase la salida de la ejecución.

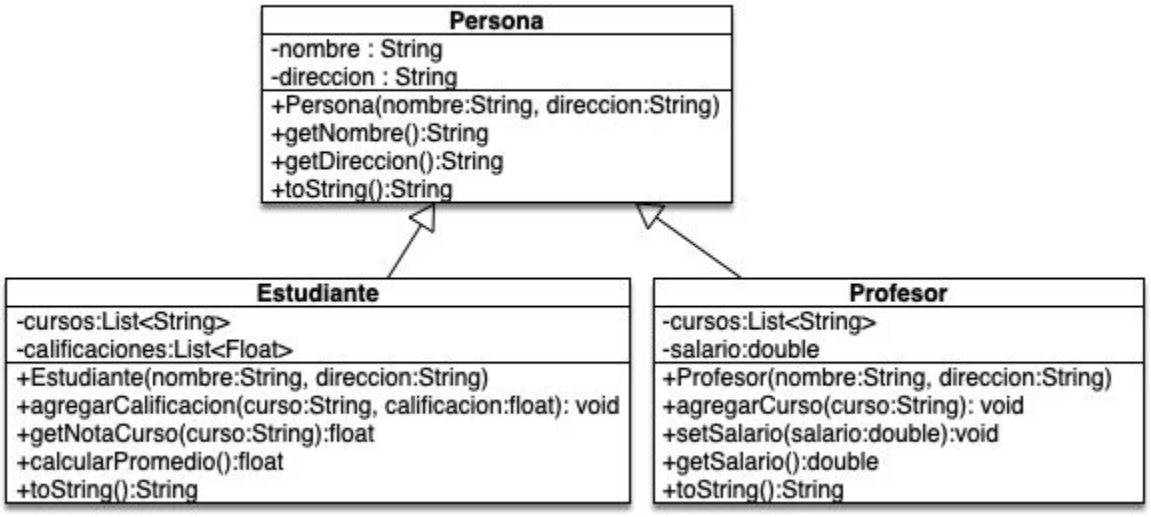
Cambiar la línea 1 por `public class Demostracion{`	
Cambiar la línea 8 por `public static void MAIN(String [] args){`	
Insertar una nueva línea debajo de la línea 10 que diga `Ejemplo.mostrarReglas();`	
Cambiar la línea 2 a public static void imprimirMensaje(){	
Cambiar la línea 2 a `public static void mostrarMensaje(){` y cambiar las líneas 10 y 11 a `Ejemplo.mostrarMensaje();`	
Reemplazar las líneas 3-4 con `System.out.println("La primera regla del club de Java es, ");`	

Se responderán solamente consultas técnicas sobre los ejercicios. Mandar mail a certificacion111mil@produccion.gob.ar con el asunto: "Consulta Ejercicios Certificación D2018"

Solución:

Cambiar la línea 1 por `public class Demostracion{`	**Error**
Cambiar la línea 8 por `public static void MAIN(String [] args){`	**Error**
Insertar una nueva línea debajo de la línea 10 que diga `Ejemplo.mostrarReglas();`	**Salida**
Cambiar la línea 2 a public static void imprimirMensaje(){	**Error**
Cambiar la línea 2 a `public static void mostrarMensaje(){` y cambiar las líneas 10 y 11 a `Ejemplo.mostrarMensaje();`	**Nada**
Reemplazar las líneas 3-4 con `System.out.println("La primera regla del club de Java es, ");`	**Salida**

7. **Implemente las clases y los métodos Java que se describen en el diagrama de clases teniendo en cuenta los siguientes detalles:**
   - El método toString de la clase Persona debe retornar un String con el nombre de la persona seguida de su dirección entre paréntesis. Por ejemplo, “Ana(Maipu 1827)”
   - El método toString de la clase Estudiante debe anteponer la cadena Estudiante: al nombre seguida de la dirección entre paréntesis. Por ejemplo, “Estudiante: Maria(San Martin 8745)”. Similarmente, el método toString de la clase Profesor debe hacerlo con la cadena Profesor:
   - No es necesario verificar por cursos o calificaciones ya agregadas en los métodos agregarCalificacion y agregarCurso
   - El método getNotaCurso debe retornar -1.0 en caso de que no se haya ingresado calificación para el curso pasado por parámetros.

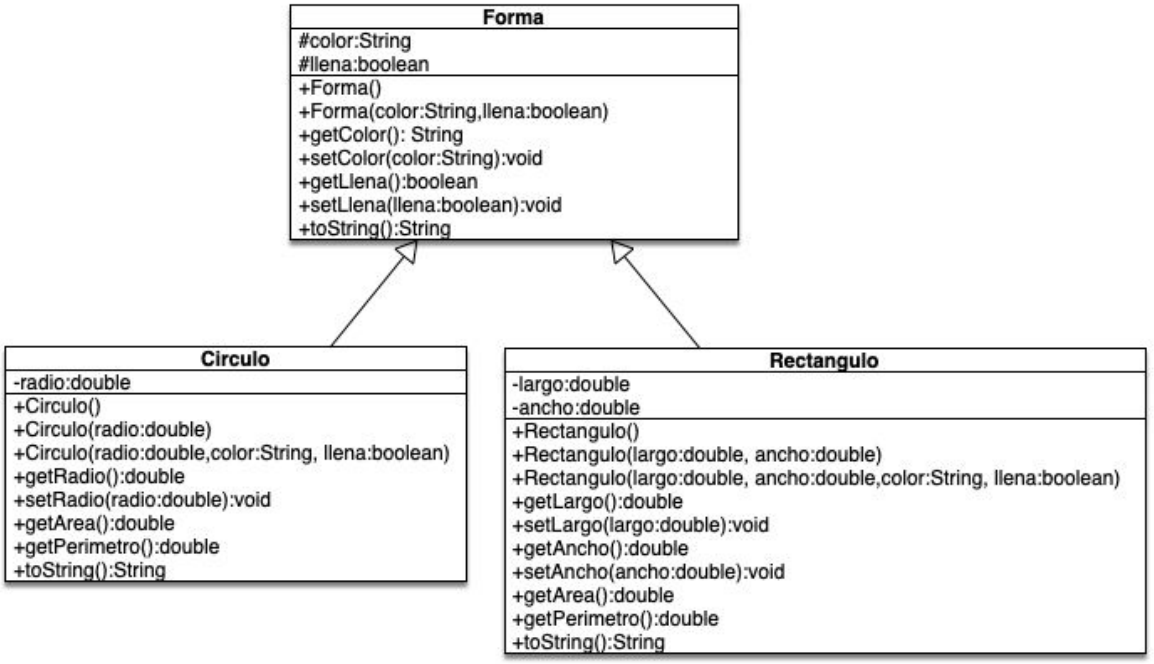


Solución:

```
public class Persona{
    private String nombre;
    private String direccion;

    public Persona(String nombre, String direccion){
        this.nombre = nombre;
        this.direccion = direccion;
    }

    public String getNombre(){
        return nombre;
    }

    public String getDireccion(){
        return direccion;
    }

    @Override
    public String toString(){
        return nombre + "(" + direccion + ")";
    }

}

public class Estudiante extends Persona{
    private List<String> cursos;
    private List<Float> calificaciones;

    public Estudiante(String nombre, String direccion){
        super(nombre, direccion);
        cursos=new ArrayList<>();
        calificaciones=new ArrayList<>();
    }

    @Override
    public String toString(){
        return "Estudiante: "+super.toString();
    }

    public void agregarCalificacion(String curso, float
calificacion){
        cursos.add(curso);
        calificaciones.add(calificacion);
    }

    public float getNotaCurso(String curso){
```

```
        Float nota=-1f;
        if(cursos.contains(curso)){
            nota=calificaciones.get(cursos.indexOf(curso));
        }
        return nota;
    }

    public float calcularPromedio(){
        Float sum=0f;
        Iterator<Float> it=calificaciones.iterator();
        while (it.hasNext()){
            Float calificacion = it.next();
            sum=sum+calificacion;
        }
        return sum/cursos.size();
    }
}

public class Profesor extends Persona{
    private List<String> cursos;
    private double salario;
    public Profesor(String nombre, String direccion){
        super(nombre, direccion);
        cursos=new ArrayList<>();
    }

    @Override
    public String toString(){
        return "Profesor: "+super.toString();
    }

    public void agregarCurso(String curso){
        cursos.add(curso);
    }

    public void setSalario(double salario){
        this.salario = salario;
    }

    public double getSalario(){
        return salario;
    }

}
```

8. **Se está trabajando en una aplicación para visualizar figuras geométricas. Implemente las clases y los métodos Java que se describen en el diagrama de clases teniendo en cuenta los siguientes detalles:**
   - La clase Forma tiene dos constructores: uno sin argumentos que inicializa color a “verde” y llena a true; y un constructor que inicializa los atributos con los valores dados.
   - El método toString de la clase Forma retorna “Forma[color=(color)]”. Dónde (color) será el almacenado en la variable de instancia.
   - La clase Circulo tiene tres constructores. El constructor sin parámetros inicializa radio en 1.
   - Recuerde que el área de un círculo es igual a □*radio^2 y el perímetro como 2*□*radio. El valor de □ puede obtener como Math.PI del paquete java.lang.
   - El método toString de la clase Circulo retorna “Un círculo de radio (radio), el cual es una sub-clase de (retorno del método toString() de la superclase)”
   - La clase Rectangulo tiene tres constructores. El constructor sin parámetros inicializa ancho y largo en 1.
   - Recuerde que el área de un rectángulo se calcula como largo*ancho y el perímetro como largo*2+ancho*2
   - El método toString de la clase Rectangulo retorna “Un rectangulo de largo (largo) y ancho (ancho), el cual es una sub-clase de (retorno del método toString() de la superclase)”



Solución:

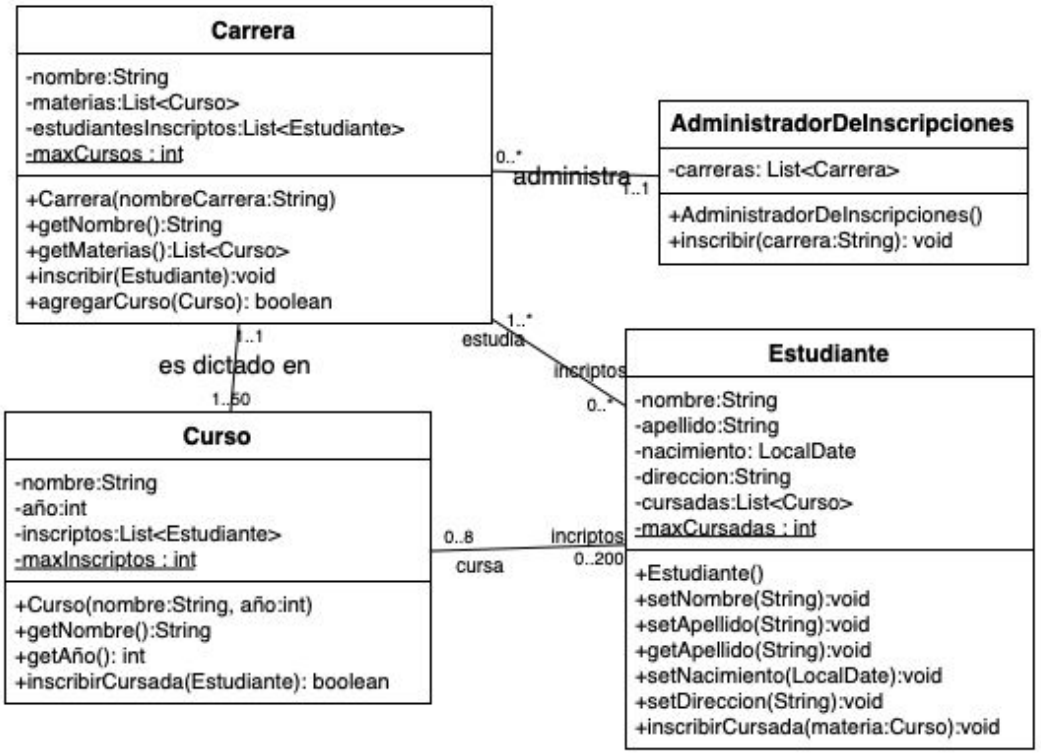
```
public class Forma{
```

```
        protected String color;
        protected boolean llena;

        public Forma(){
                color = "verde";
                llena = true;
        }
        public Forma(String color,boolean llena){
                this.color = color;
                this.llena = llena;
        }

        @Override
        public String toString(){
                return "Forma[color=" + color + "]";
        }

        public void setColor(String color){
                this.color = color;
        }
        public String getColor(){
                return color;
        }

        public boolean getLlena(){
                return llena;
        }
        public void serLlena(boolean llena){
                this.llena = llena;
        }
}

public class Circulo extends Forma{

        private double radio;

        public Circulo(double radio,String color,boolean llena){
                super(color,llena);
                this.radio = radio;
        }

        public Circulo(double radio){
                super();
                this.radio = radio;
        }

        public Circulo(){
                super();
                this.radio = 1;
```

```
	}

	public double getRadio(){
		return radio;
	}

	@Override
	public String toString(){
		return "Un circulo de radio"+radio+", el cual es una
sub-clase de "+super.toString();
	}

	public double getArea(){
		return Math.PI*radio*radio;
	}

	public double getPerimetro(){
		return 2*Math.PI*radio;
	}
}

public class Rectangulo extends Forma{
	private double largo;
	private double ancho;

	public Rectangulo(double largo, double ancho, String
color,boolean llena){
		super(color,llena);
		this.largo = largo;
		this.ancho = ancho;
	}
	public Rectangulo(double largo, double ancho){
		super();
		this.largo = largo;
		this.ancho = ancho;
	}
	public Rectangulo(){
		super();
		this.largo = 1;
		this.ancho = 1;
	}
	public String toString(){
		return "Un Rectangulo de largo " + largo + "y ancho
" + ancho + " que es una sub-clase de " + super.toString();
	}

	public double getArea(){
		return largo*ancho;
	}
```

```
        public double getPerimetro(){
                return largo*2 + ancho*2;
        }

        public void setLargo(double largo){
                this.largo = largo;
        }
        public double getLargo(){
                return largo;
        }

        public void setAncho(double ancho){
                this.ancho = ancho;
        }
        public double getAncho(){
                return ancho;
        }

}
```

9. **Implemente las clases y los métodos Java que se describen en el diagrama de clases teniendo en cuenta los siguientes detalles:**
   - El método agregarCurso solo agregará el curso pasado por parámetros si se cumple la restricción indicada en la multiplicidad. En caso de poder agregarse satisfactoriamente devuelve true.
   - Idem método inscribirCursada de la clase Curso.
   - El método inscribirCursada de la clase Estudiante deberá invocar al método inscribirCursada de la clase Curso y sólo inscribirá el curso pasado por parámetros si el resultado del método invocado es true y si se cumple la restricción indicada en la multiplicidad.



Solución:

```
public class AdministradorDeInscripciones{
    private List<Carrera> carreras;
    public AdministradorDeInscripciones(){
       carreras=new ArrayList<>();
    }
    public void inscribir(String carrera){
        carreras.add(new Carrera(carrera));
    }
}

public class Carrera{
    private String nombre;
    private List<Curso> materias;
    private List<Estudiante> estudiantesInscriptos;
    private static int maxCursos = 50;
    public Carrera(String nombreCarrera){
        nombre=nombreCarrera;
        materias=new ArrayList<>();
        estudiantesInscriptos=new ArrayList<>();
    }

    public String getNombre(){
        return nombre;
    }
```

Se responderán solamente consultas técnicas sobre los ejercicios. Mandar mail a certificacion111mil@produccion.gob.ar con el asunto: "Consulta Ejercicios Certificación D2018"

```
    public List<Curso> getMaterias(){
        return materias;
    }

    public void inscribir(Estudiante e){
        estudiantesInscriptos.add(e);
    }

    public boolean agregarCurso(Curso c){
        if(materias.size()<maxCursos){
            materias.add(c);
            return true;
        }
        return false;
    }

}

public class Curso{
    private String nombre;
    private int año;
    private List<Estudiante> inscriptos;
    private static int maxInscriptos = 200;


    public Curso(String nombre, int año){
       this.nombre = nombre;
       this.año = año;
       inscriptos=new ArrayList<>();
    }

    public String getNombre(){
        return nombre;
    }

    public int getAño(){
        return año;
    }

    public boolean inscribirCursada(Estudiante e){
        if(inscriptos.size()<maxInscriptos){
            inscriptos.add(e);
            return true;
        }
        return false;
    }

}
```

```
public class Estudiante{
    private String nombre;
    private String apellido;
    private LocalDate nacimiento;
    private String direccion;
    private List<Curso> cursadas;
    private static int maxCursadas = 8;


    public Estudiante(){
       cursadas=new ArrayList<>();
    }

    public void setNombre(String nombre){
        this.nombre = nombre;
    }

    public void setApellido(String apellido){
        this.apellido = apellido;
    }

    public String getApellido(){
        return apellido;
    }

    public void setNacimiento(LocalDate nacimiento){
        this.nacimiento = nacimiento;
    }

    public void setDireccion(String direccion){
        this.direccion = direccion;
    }

    public void inscribirCursada(Curso materia){
            materia.inscribirCursada(this);
            if(cursadas.size()<maxCursadas &&
materia.inscribirCursada(this)){
                cursadas.add(materia);
            }
    }
}
```

9.1. Es necesario incorporar un método para obtener los cursos de todas las carreras de un año en particular dado.

a.) Indique en qué clase/s debería definirse esa nueva funcionalidad.
b.) Implemente el/los método/s necesarios en Java. Si considera necesario nuevas variables defínalas y diga en qué clase se implementan.

Solución:

OPCIÓN 1

a) AdministradorDeInscripciones

b)

```
public List<Curso> obtenerCursosPorAño(int año){
        List<Curso> ret=new ArrayList<>();
        for (Iterator<Carrera> iterator = carreras.iterator();
iterator.hasNext();){
            Carrera carrera = iterator.next();
            List<Curso> materias =carrera.getMaterias();
            for (Iterator<Curso> iterator1 = materias.iterator();
iterator1.hasNext();){
                Curso curso = iterator1.next();
                if(curso.getAño()==año){
                    ret.add(curso);
                }
            }
        }
        return ret;
    }
```

OPCIÓN 2

a) AdministradorDeInscripciones, Carrera, Curso

b)

En AdministradorDeInscripciones

```
   public List<Curso> obtenerCursosPorAño(int año){
        List<Curso> cursosSegunAño=new ArrayList<>();
        Iterator<Carrera> iterator=carreras.iterator();
        while (iterator.hasNext()){
            Carrera carrera = iterator.next();
            cursosSegunAño.addAll(carrera.obtenerCursosPorAño(año));
        }
        return cursosSegunAño;
    }
```

En Carrera

```
  public List<Curso> obtenerCursosPorAño(int año){
        List<Curso> cursosDelAño=new ArrayList<>();
        Iterator<Curso> it=materias.iterator();
```

```
        while (it.hasNext()){
            Curso curso = it.next();
            if(curso.esDelAño(año))
               cursosDelAño.add(curso);
        }
        return cursosDelAño;
    }
```

En Curso

```
  public boolean esDelAño(int año){
        return this.año==año;
    }
```

9.2. Otro desarrollador implementó el siguiente método en la clase Carrera. El método debería devolver todos los estudiantes cuyo apellido coincida por el pasado por parámetros. Sin embargo, no funciona. Encuentre el error.

```
public List<Estudiante> getEstudiantesPorApellido(String apellido){
        List<Estudiante> estudiantesConApellido=new ArrayList<>();

        for (Iterator<Estudiante> iterator =
estudiantesInscriptos.iterator(); iterator.hasNext();){
            Estudiante estudiante = iterator.next();
            if(estudiante.getApellido()==apellido)
                estudiantesConApellido.add(estudiante);
        }

        return estudiantesConApellido;
    }
```

Solución:

```
public List<Estudiante> getEstudiantesPorApellido(String apellido){
        List<Estudiante> estudiantesConApellido=new ArrayList<>();

        for (Iterator<Estudiante> iterator =
estudiantesInscriptos.iterator(); iterator.hasNext();){
            Estudiante estudiante = iterator.next();
            if(estudiante.getApellido().equals(apellido))
                estudiantesConApellido.add(estudiante);
        }

        return estudiantesConApellido;
    }
```

9.3. Otro desarrollador implementó el siguiente método en la clase Carrera pero no le puso un nombre representativo. Explique qué hace el método.

```
private void ???????(){
  int index=0;
  boolean intercambio = true;
  Curso auxiliar;
  while (intercambio) {
    intercambio = false;
    index++;
    for (int i = 0; i < materias.size() - index; i++) {
      if (materias.get(i).getAño() > materias.get(i + 1).getAño()) {
        auxiliar = materias.get(i);
        materias.add(i, materias.get(i + 1));//Agrega el valor pasado en el
segundo parámetro en la posición i de la lista. El valor que se
encontraba en la posición i se mueve a la derecha
        materias.remove(i+1);
        materias.add(i+1, auxiliar);
        materias.remove(i+2);
        intercambio = true;
    }
  }
}
```

Solución:

El método ordena los cursos según su año.

10. Escriba la documentación del método inscribirCursada de la clase Curso del ejercicio 10.

Solución:

```
/**El metodo inscribe al estudiante pasado por parametros en el curso.
    * El estudiante es inscripto solo si no se completo el cupo del curso (200
alumnos).
    * En caso de que el curso este completo retorna false
    * @param e Estudiante a inscribir
    * @return true si el estudiante es inscripto en el curso
    */
```

11. Escriba la documentación del método getEstudiantesPorApellido que corrigió en el ejercicio 10.2.

Solución:

```
/** El metodo retorna una lista con los estudiantes que tienen el apellido pasado por
    *parametros
    *
```

```
    * @param apellido de los alumnos a retornar
    * @return lista de estudiantes con el apellido pasado por parametro
    */
```

12. Dadas las siguientes clases, analice el comportamiento en común. A partir del análisis implemente una interface que lo abstraiga.

```
public class Gato{
    public Gato(){

    }

    public void comer(){
          System.out.println("El gato come plancton.");
     }

    public void jugar(){
        System.out.println("El gato juega");
    }

}


public class Pez{
    public Pez(){

    }

    public void comer(){
         System.out.println("El pez come plancton.");
    }

    public void jugar(){
         System.out.println("El PEZ juega");
    }
}
```

Solución:

```
public interface Animal{

    void comer();

    void jugar();

}
```

```
public class Gato implements Animal{
    public Gato(){

    }

    @Override
    public void comer(){
          System.out.println("El gato come plancton.");
     }

    @Override
    public void jugar(){
        System.out.println("El gato juega");
    }

}


public class Pez implements Animal{
    public Pez(){

 }
 @Override
public void comer(){
   System.out.println("El pez come plancton.");
 }

 @Override
 public void jugar(){
   System.out.println("El PEZ juega");
  }

}
```