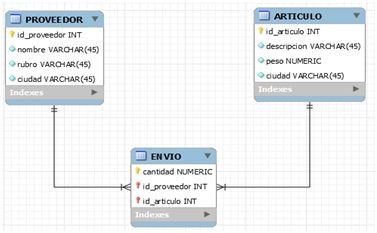
**Ejercicios de Repaso para la**

**Certificación**

Ejercicios de Bases de Datos

**1. Dado el siguiente de Diagrama de Entidad-Relación y la consulta SQL. Modifique la consulta SQL ya que contiene errores y describa el resultado de la consulta.**



SELECT id\_proveedor, id\_articulo

FROM Proveedor p JOIN Envio e ON p.nombre = e.id\_proveedor

Error 1: JOIN no corresponde comparar nombre con id\_proveedor

Error 2: Debemos anteponer el alias en el select cuando el nombre del campo se repite en varias tablas

SELECT p.id\_proveedor, e.id\_articulo

FROM Proveedor p JOIN Envio e ON p.id\_proveedor = e.id\_proveedor

Una version mejorada:

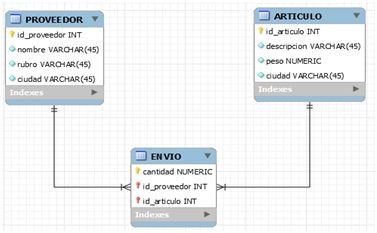
SELECT p.id\_proveedor, p.nombre, e.id\_articulo, a.descripcion

FROM Proveedor p JOIN Envio e ON p.id\_proveedor = e.id\_proveedor

JOIN Articulo a ON a.id\_articulo = e.id\_articulo

Esta consulta lista los articulos que se encargaron a los distintos proveedores.

**2. Dado el siguiente de Diagrama de Entidad-Relación y la consulta SQL. Modifique la consulta SQL ya que contiene errores y describa el resultado de la consulta.**



SELECT SUM (e.cantidad), e.id\_proveedor, 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)

Consulta corregida:

SELECT SUM (e.cantidad), e.id\_proveedor, 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)

Error #1: El AND se reemplaza por el WHERE

Error #2: En la sub consulta del IN devolver id\_articulo en vez de descripcion

Error #3: Reemplazamos el SUM por COUNT para calcular la cantidad de veces que un proveedor envio un determinado articulo

Consulta mejorada:

SELECT COUNT (e.cantidad), e.id\_proveedor, e.id\_articulo

FROM Proveedor p JOIN Envio e ON p.id\_proveedor = e.id\_proveedor

WHERE e.id\_articulo IN (SELECT a.id\_articulo FROM Articulo a)

Esta consulta calcula la cantidad de veces que un proveedor envio un determinado articulo

**3. Dada la siguiente consulta SQL y las siguientes tuplas de ejemplo, determinar el resultado de la consulta:**

SELECT SUM (e.cantidad), e.id\_proveedor, 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) HAVING e.cantidad > 100

GROUP BY e.id\_proveedor

ORDER BY e.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, “producto X”, “limpieza”, “Tandil”

2, “producto Y”, “higiene”, “Azul”

3, “producto Z”, “farmacia”, “Bolivar”

4, “producto W”, “limpieza”, “Mar del Plata”

ENVIO:

1,1,120

1,2,160

3,1,90

4,3,100

Consulta corregida:

SELECT COUNT (e.cantidad), e.id\_proveedor, e.id\_articulo

FROM Proveedor p JOIN Envio e ON p.id\_proveedor = e.id\_proveedor

WHERE e.id\_articulo IN (SELECT a.id\_articulo FROM Articulo a)

GROUP BY e.id\_proveedor, e.id\_articulo, p.ciudad

HAVING COUNT (e.cantidad) > 100

ORDER BY p.ciudad DESC

Error #1: El SUM se reemplaza por el COUNT

Error #2: El AND se reemplaza por el WHERE

Error #3: El HAVING siempre va despues del GROUP BY

Error #4: En el ORDER BY no se puede usar e porque ENVIO no tiene Ciudad

Retorna la cantidad de veces que un proveedor envio un determinado articulo,

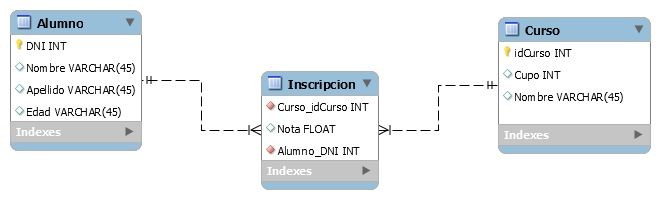
siempre y cuando la cantidad es mayor a cien y ordena por la ciudad del proveedor descendentemente.

Los datos de la tabla ENVIO son invalidos ya que no existen ARTICULOS con esos valores.

Ademas los valores de CANTIDAD estan repetidos.

**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. A partir del DER del ejercicio anterior, el equipo necesita 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 necesita que escribas la consulta SQL correspondiente.**

.



SELECT a.Nombre, a.Apellido, a.DNI, a.Edad, c.Nombre

FROM Alumno a

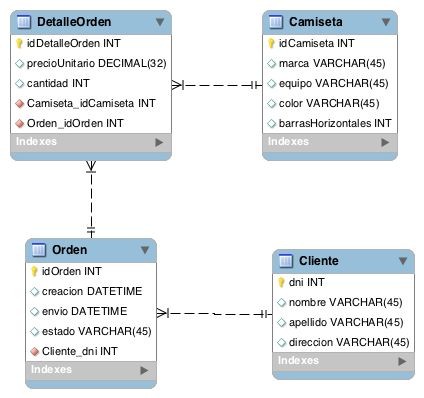
JOIN Inscripcion i ON i.Alumno\_DNI = a.DNI

JOIN Curso c ON c.idCurso = i.Curso\_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.**



SELECT o.idOrden

FROM Orden o

JOIN Cliente c ON o.Cliente\_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**

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

)

Comentarios:

Define una relacion entre Infraccion y ActaConstatacion a traves de las columnas

numero en ActaConstatacion y numeroActaConstatacion en Infraccion-

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

<column name="monto" />

</property>

BigDecimal pertenece al paquete java.math

Date pertenece al paquete java utils

**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**

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

)

Comentarios:

Infraccion tiene un atributo del tipo InfraccionNomenclada

<many-to-one name="infraccionNomenclada" class="InfraccionNomenclada" column="codigoInfraccionNomenclada" cascade="none" not-null="true" insert="true" update="false" lazy="false" />

Ejercicios de Java

**1. Considere el siguiente método**

public void ifElseMisterioso(int x, int y) {

int z = 4;

if (z <= x) {

z = x + 1;

} else {

z = z + 9;

}

if (z <= y) {

y++;

}

System.out.println(z + " " + y);

}

**Para cada una de las siguientes invocaciones, determinar la salida que se obtiene**:

|  |  |
| --- | --- |
| ifElseMisterioso(3,20); | ’13, 21’ |
| ifElseMisterioso(4,5); | ’5, 6’ |
| ifElseMisterioso(5,5); | ’6, 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)**

int[] a1 = {26, 19, 14, 11, 10};

**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]);

int [] 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;

}

}

* La declaración de x e y dentro del constructor
* Sobra la palabra void en la firma del metodo

**5. Considere las siguientes invocaciones. Determine la salida que se produce.**

public class Raro{

public static void main(String[] args) { Raro.primero();

Raro.tercero(); Raro.segundo();

Raro.tercero();

}

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

}

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

}

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

segundo();

}

}

Consola:

“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 produciría que el programa no compile o que de un error durante su ejecución.

● “Salida”. Si cambiaría la salida de la ejecución.

|  |  |
| --- | --- |
| 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. Dada la siguiente clase:**

public class ListaDeProductos{

private List<Productos> productos;

public List<Produtos> getProductos(){

return productos;

}

}

**Cree una ListaDeProductosFiltrada que retorne los productos que empiezan con una letra dada (para ello, puede heredar de la clase dada y extender su funcionalidad).**

public class ListaDeProductosFiltrada extends ListaDeProducto {

public ListaDeProductosFiltrada(char letra) {

this.letra = letra;

}

private char letra;

@Override

public List<Producto> getProductos() {

List<Producto> productosFiltrados = new ArrayList<Producto>();

for (Producto producto : super.getProductos() {

if (producto.startsWith(letra)) {

productosFiltrados.add(producto);

}

}

return productosFiltrados;

}

}

**8. Extienda el funcionamiento de la siguiente clase, agregando un chequeo por usuario (si el nombre de usuario que intenta abrir el correo no coincide con el nombre de usuario configurado, imprimir un error):**

public class CorreoPublico { String contenido;

public String leer(String nombreUsuario) {

return contenido;

}

}

public class CorreoExtendido extends CorreoPublico {

private String nombreUsuario;

@Override

public String leer(String nombreUsuario) {

if (this.nombreUsuario.equals(nombreUsuario)) {

return this.super();

}

System.out.println("Error");

return "";

}

}

**9. Realice las modificaciones necesarias sobre Noticia que permitan extenderla. El objetivo es crear una subclase llamada NoticiaResumida que solo muestra los primeros 50 caracteres del contenido más “...” (3 puntos suspensivos) que indican que el contenido continúa. Su título también es limitado a 20 caracteres, pero no se colocan puntos.**

public class Noticia{

private String titulo;

private String contenido;

public String getTitulo() {

return titulo;

}

public String getContenido() {

return contenido;

}

// Retorna el contenido con la primer letra capital. protected String getContenidoConFormato() {

return contenido.substring(0,1).toUpperCase()

+ contenido.substring(1);

}

// Retorna el título todo en mayúsculas protected String getTituloConFormato() { return titulo.toUpperCase();

}

public String imprimir(){

return getTituloConFormato() + "\n"

+ getContenidoConFormato();

}

}

public class NoticiaResumida extends Noticia {

@Override

public String getContenidoConFormato() {

return super.getContenidoConFormato().substring(0, 50) + '...';

}

@Override

public String getTituloConFormato() {

return super.getTituloConFormato().substring(0, 20);

}

}

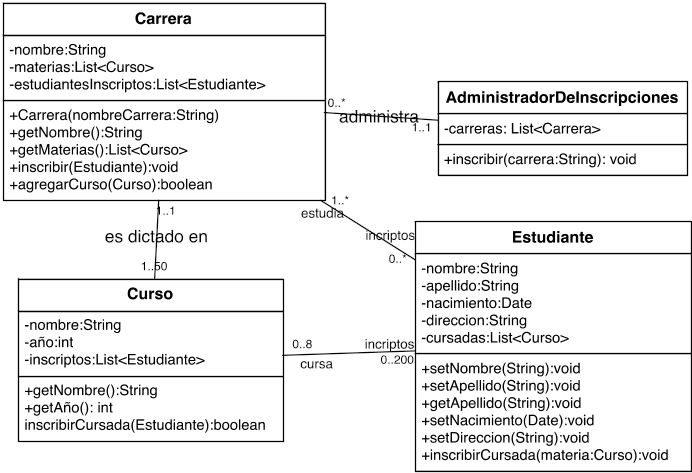
**10. Implemente las clases y los métodos Java que se describen en el diagrama de clases teniendo en cuenta los siguientes detalles:**

● 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.



public class Carrera() {

public boolean agregarCurso(Curso curso) {

if (this.materias.size() >= 50) {

return false;

}

materias.add(curso);

return true;

}

}

public class Curso() {

public boolean inscribirCursada(Estudiante estudiante) {

if (this.inscriptos.size() >= 200) {

return false;

}

inscriptos.add(estudiante):

return true;

}

}

public class Estudiante() {

public void InscribirCursada(Curso materia) {

if (this.cursadas.size() <= 8 && materia.inscribirCursada(this)) {

this.cursadas.add(materia);

}

}

}

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

1. Indiqué en qué clase/s debería definirse esa nueva funcionalidad.

AdministradorDeInscripciones

b. ) Implemente el/los método/s necesarios en Java. Si considera necesario nuevas variables defínalas y diga en qué clase se implementan.

public class AdministradorDeInscripciones() {

public List<Curso> getMateriasPorAnio(int anio) {

List<Curso> materias = new ArrayList<Curso>();

for(Carrera carrera : this.carreras) {

for(Curso curso : carrera.getMaterias()) {

if (curso.getAnio() == anio) {

materias.add(curso);

}

}

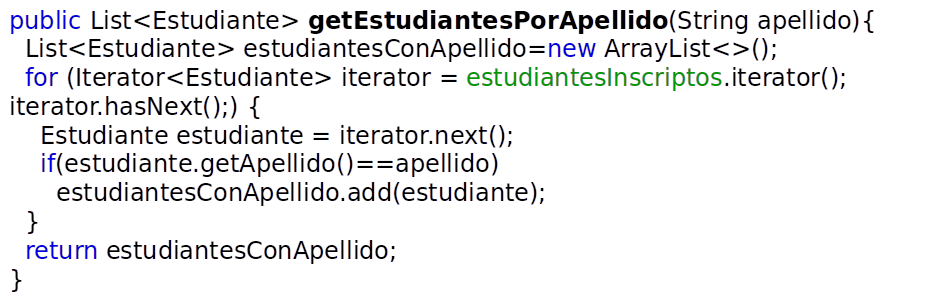
}

return materias;

}

}

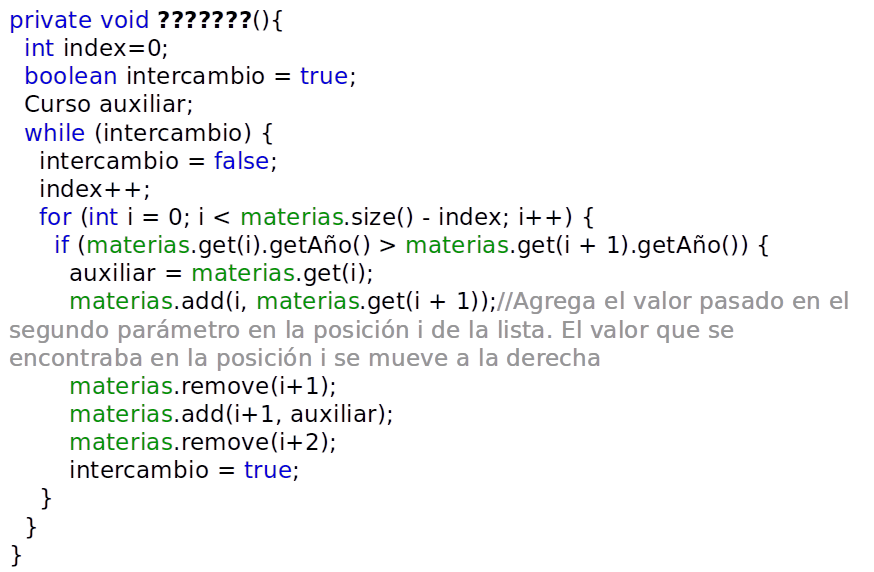
**10.2. Otro desarrollador implementó el siguiente método en la clase Carrera. El método debería devolver todos los estudiantes cuyo apellido coincida por el pasado por parámetros. Sin embargo, no funciona. Encuentre el error.**



La comparación entre estudiante.getApellido() y apellido tiene que ser usando

equals en vez de ==, seria: estudiante.getApellido().equals(apellido)

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



El método ordena las materias según el año en forma ascendente.

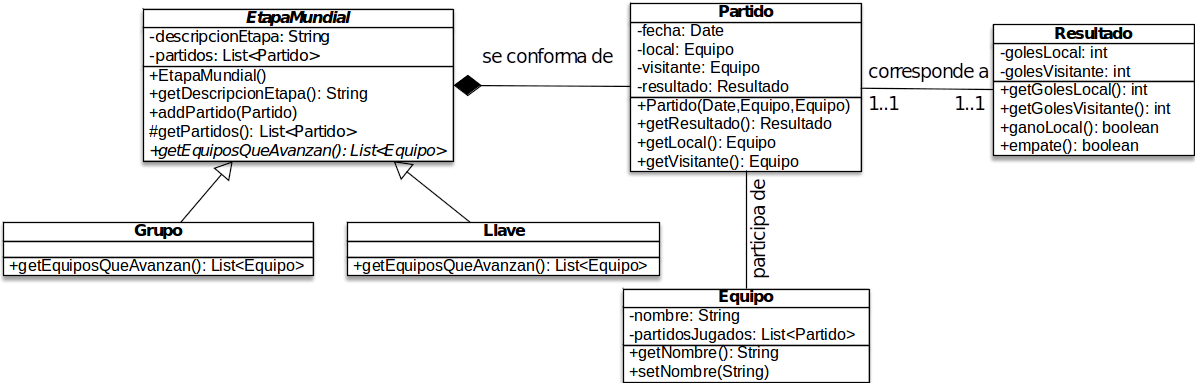
**11. Implemente las clases y los métodos Java que se describen en el diagrama de clases teniendo en cuenta los siguientes detalles:**

● Un grupo de un mundial está compuesto de 4 equipos. Solo los dos con mayor cantidad de puntos avanzan a la siguiente rueda. Ganador=3 puntos. Empate=1 punto. Suponga que no hay equipos con la misma cantidad de puntos una vez jugados todos los partidos del grupo En las llaves (ej. octavos, cuartos, etc.) avanzan los ganadores (los partidos no pueden terminar en empate).

● Note que el método getEquiposQueAvanzan() de la clase EtapaMundial es abstracto.

● El constructor de EtapaMundial debe inicializar la lista partidos.

● Puede crear métodos si lo considera necesario.



Ver proyecto p111mil.Mundial en Repositorio

public class Llave extends EtapaMundial {

@Override

public List<Equipo> getEquiposQueAvanzan() {

List<Equipo> equipos = new ArrayList<Equipo>();

if (this.getPartidos().size() > 0) {

Partido partido = this.getPartidos().get(0);

if (partido.getResultado().ganoLocal()) {

equipos.add(partido.getLocal());

} else {

equipos.add(partido.getVisitante());

}

}

return equipos;

}

}

public class Grupo extends EtapaMundial {

// Listas paralelas para armar la tabla de posiciones

List<Equipo> equiposGrupo;

List<Integer> puntosEquipos;

/\*\*

\* Calcula la tabla de posiciones y retorna los dos equipos clasificados

\* @return Lista de equipos

\*/

@Override

public List<Equipo> getEquiposQueAvanzan() {

// Inicializo las listas

equiposGrupo = new ArrayList<Equipo>();

puntosEquipos = new ArrayList<Integer>();

// Recorro todos los partidos para generar una lista con todos los equipos

for(Partido partido : this.getPartidos()) {

addEquipoEnListaSiNoExiste(partido.getLocal());

addEquipoEnListaSiNoExiste(partido.getVisitante());

}

// Recorro los equipos y sumo en la lista paralela los puntos segun el resultado

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

for(Partido partido : this.getPartidos()) {

if (equiposGrupo.get(i).getNombre().equals(partido.getLocal().getNombre())) {

if (partido.getResultado().ganoLocal()) {

puntosEquipos.set(i, puntosEquipos.get(i) + 3);

} else if (partido.getResultado().empate()) {

puntosEquipos.set(i, puntosEquipos.get(i) + 1);

}

} else if (equiposGrupo.get(i).getNombre().equals(partido.getVisitante().getNombre())) {

if (partido.getResultado().empate()) {

puntosEquipos.set(i, puntosEquipos.get(i) + 1);

} else if (! partido.getResultado().ganoLocal()) {

puntosEquipos.set(i, puntosEquipos.get(i) + 3);

}

}

}

}

ordenarPosiciones();

// Retornamos los primeros dos equipos segun las posiciones

return equiposGrupo.subList(0, 2);

}

/\*\*

\* Se fija si el equipo no fue todavia agregado en la lista interna y de ser asi lo agrega.

\* Si el equipo ya existe no hace nada

\* @param equipoParaAgregar

\*/

private void addEquipoEnListaSiNoExiste(Equipo equipoParaAgregar) {

for(Equipo equipoExistente : equiposGrupo) {

if (equipoExistente.getNombre().equals(equipoParaAgregar.getNombre())) {

return;

}

}

equiposGrupo.add(equipoParaAgregar);

puntosEquipos.add(0);

}

/\*\*

\* Ordena las listas paralelas de puntos y de equipos segun los puntos en forma descendente

\*/

private void ordenarPosiciones() {

for(int j = 0; j < puntosEquipos.size(); j++){

for(int i = j + 1; i < puntosEquipos.size(); i++){

if(puntosEquipos.get(i) > puntosEquipos.get(j)){

int puntos = puntosEquipos.get(j);

puntosEquipos.set(j, puntosEquipos.get(i));

puntosEquipos.set(i, puntos);

Equipo equipo = equiposGrupo.get(j);

equiposGrupo.set(j, equiposGrupo.get(i));

equiposGrupo.set(i, equipo);

}

}

}

}

}

**11.1. Para realizar estadísticas, se necesita saber la diferencia de goles de un equipo. Por ejemplo, si en los partidos disputados en el mundial por el equipo A recibió 3 goles y convirtió 10, entonces, la diferencia de goles será 7.**

1. Indique en qué clase se implementará dicha funcionalidad.

En la clase Equipo.

1. Implemente en Java el/los método/s correspondientes. Si considera necesario nuevas variables defínalas y diga en qué clase se implementan.

public int getDiferenciaDeGoles() {

int golesFavor = 0;

int golesContra = 0;

for(Partido partido : this.partidosJugados) {

if (partido.getLocal().nombre.equals(this.nombre)) {

golesFavor = golesFavor + partido.getResultado().getGolesLocal();

golesContra = golesContra + partido.getResultado().getGolesVisitante();

} else {

golesFavor = golesFavor + partido.getResultado().getGolesVisitante();

golesContra = golesContra + partido.getResultado().getGolesLocal();

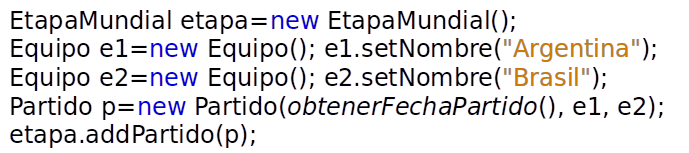
}

}

return golesFavor - golesContra;

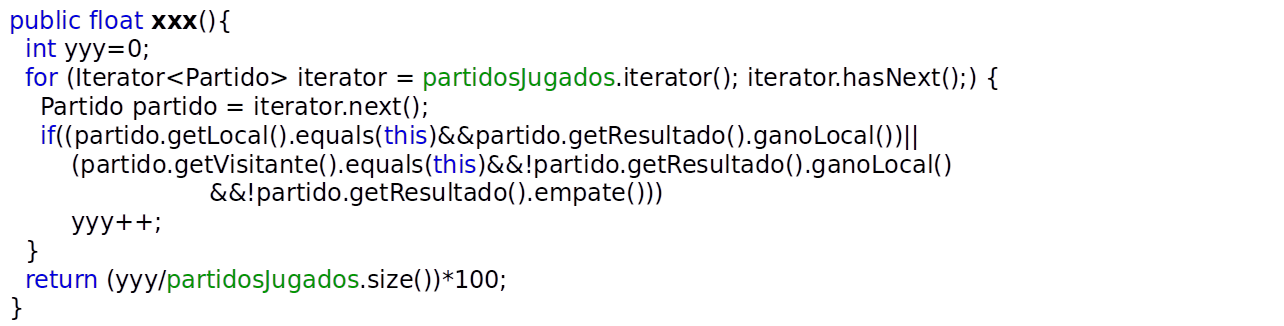
}

**11.2. El siguiente código no compila. Explique a qué se debe. Suponga que el método obtenerFechaPartido() se encuentra implementado y que devuelve un objeto de tipo Date válido.**



El código no compila porque EtapaMundial es una clase abstracta

**11.3. Un desarrollador implementó el siguiente método en la clase Equipo pero no usó nombres representativos. Explique qué hace el método.**



Calcula el porcentaje de partidos ganados

**12. En base al siguiente método de la clase Licencia escriba la documentación del método en formato JavaDoc:**

public BigDecimal cuantoDebePorInfraccionesNoPagadas(LocalDate fechaDesde, LocalDate fechaHasta) {

BigDecimal total = BigDecimal.ZERO;

// iteramos sobre las actas de constatación asociadas a la licencia del conductor

Iterator<ActaConstatacion> iter = actas.iterator();

while (iter.hasNext()) {

// obtenemos el acta actual

ActaConstatacion acta = iter.next();

// comprobamos que la fecha sea entre las buscadas

if (acta.estasEnPeriodo(fechaDesde, fechaHasta) && !acta.estaPagada()) {

// acumulamos la cantidad de infracciones

total = total.add(acta.calcularTotalInfracciones());

}

}

return total;

}

/\*\*

\* Calcula la deuda por infracciones no pagadas

\* @param fechaDesde

\* @param fechaHasta

\* @return el total de dinero adeudado

\*/

public BigDecimal cuantoDebePorInfraccionesNoPagadas(LocalDate fechaDesde,

LocalDate fechaHasta) {

…

}

**13. En base al siguiente método de la clase Licencia escriba la documentación del método en formato JavaDoc:**

public int cuantasInfraccionesEnPeriodo(LocalDate fechaDesde, LocalDate fechaHasta) {

int infracciones = 0;

// iteramos sobre las actas de constatación de la licencia del conductor

Iterator<ActaConstatacion> iter = actas.iterator();

while (iter.hasNext()) {

// obtenemos el acta actual

ActaConstatacion acta = iter.next();

// comprobamos que la fecha sea entre las buscadas if (acta.estasEnPeriodo(fechaDesde, fechaHasta)) {

// acumulamos la cantidad de infracciones infracciones += acta.cuantasInfraccionesContiene();

}

}

return infracciones;

}

/\*\*

\* Calcula cuantas infracciones se cometieron entre las fechas

\* @param fechaDesde

\* @param fechaHasta

\* @return la cantidad de infracciones cometidas en el periodo

\*/

public int cuantasInfraccionesEnPeriodo(LocalDate fechaDesde,

LocalDate fechaHasta) {

}

**14. Dadas las siguientes clases, analice el comportamiento en común para definir una interface que lo abstraiga.**

**public class Pez {**

**public Pez() {**

**}**

**public void comer() {**

**System.*out.println("El pez come plancton.");***

**}**

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

**}**

**}**

**public class Gato {**

**public Gato() {**

**}**

**public void comer() {**

**System.*out.println("El gato come plancton.");***

**}**

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

**}**

**}**

Como seria la interface

public interface Animal {

void comer();

void jugar();

}

Como quedaría la clase Pez

public class Pez implements Animal {

public Pez() {

}

public void comer() {

System.out.println("El pez come plancton.");

}

public void jugar() {

System.out.println("El PEZ juega");

}

}