Relación 4

Ejercicio 1. Acceso a métodos en Ruby

Sea el siguiente código en Ruby:

```
class Ejemplo
      def publico_implicito_protegido
           metodo_protegido
      end
      def publico_explicito_protegido
           self.metodo_protegido
      end
      def publico_implicito_privado
           metodo_privado
      end
      def publico_explicito_privado
           self.metodo_privado
      end


      protected
      def metodo_protegido
           puts "metodo protegido"
      end

      private
      def metodo_privado
           puts "método privado"
      end
end
```

1. Indica cuál es el resultado de ejecutar las siguientes líneas de código

a)Ejemplo.new.publico_implicito_protegido	
b)Ejemplo.new.publico_explicito_protegido	
c)Ejemplo.new.metodo_protegido	
d)Ejemplo.new.publico_implicito_privado	
e)Ejemplo.new.publico_explicito_privado	
f) Ejemplo.new.metodo_privado	

2. Explica el resultado obtenido en cada uno de estos puntos

Ejercicio 2. Visibilidad de métodos en Java

Sea el siguiente código en Java:

```
public class Ejercicio13 {

    public void publico(){
        System.out.println("método público");
    }
    void paquete(){
        System.out.println("método paquete");
    }
    protected void protegido(){
        System.out.println("método protegido");
    }
    private void privado(){
        System.out.println("método privado");
    }
}
```

Prueba el siguiente código en la misma clase, en otra clase del mismo paquete y en otra clase de distinto paquete:

```
 public static void main(String[] args) {
     Ejercicio13 e13 = new Ejercicio13();
     e13.publico();
     e13.protegido();
     e13.paquete();
     e13.privado();
}
```

Plantéate qué ocurre en cada una de las situaciones indiadas.

Ejercicio 3. Visibilidad de variables en Java

Sea el siguiente código en Java:

```
public class Ejercicio14 {
    private int i=0;
    int j= 1;
    protected int k=2;
    public int l=3;
}
```

Prueba el siguiente código en la misma clase, en otra clase del mismo paquete y en otra clase de distinto paquete:

```
 public static void main(String[] args) {
      Ejercicio14 e14 = new Ejercicio14();
      System.out.println("privada : " + e14.i);
      System.out.println("paquete : " + e14.j);
      System.out.println("protected : " + e14.k);
      System.out.println("public : " + e14.l);
}
```

Plantéate qué ocurre en cada una de las situaciones indicadas.

Ejercicio 4. Dada la siguiente clase en Java:

```
public class Vertebrado
{...
public ArrayList<String> partesDelAbdomen();
public void desplazarse();
public String comunicarse(Vertebrado vertebrado);
}
```

Indica con una cruz en la casilla correspondiente según si la declaración de los siguientes métodos de la clase Mamifero, subclase de Vertebrado, sobrecargan (overloading) o redefinen (overriding) a los de la clase Vertebrado:

	Sobrecarga / Overloading	**Redefinición / Overriding**
`public  ArrayList<String> partesDelAbdomen();`		
`public void desplazarse(Modo m);`		
`public String comunicarse(Vertebrado vertebrado);`		
`public String comunicarse(Mamifero mamifero);`		

Ejercicio 5. A partir de las siguientes clases, implementa la clase *Alumno-Informatica* que hereda de Alumno. Debe tener una nueva variable de instancia que es una colección de strings con los *dispositivos* que utiliza: como por ejemplo {PC, tablet, smartphone}. Esa variable debe poder consultarse y modificarse. También debe heredar el método *estudiar*(), pero modificando su implementación para que además de estudiar como los otros alumnos, diga que estudia con el último dispositivo de su colección. No olvides implementar su constructor. Implementa este mismo problema en Ruby.

```
public class Persona {
    protected String nombre ;
    public Persona(String nom) {  this.setNombre(nom); }
    protected String getNombre() { return this.nombre; }
    protected void setNombre(String nom) { this.nombre=nom; }
    public String hablar(){ return 'bla bla bla';}
}
public class Alumno extends Persona {
      public String  carrera;
      public int curso;

public Alumno(String nom, String carr,int cur)
{     super(nom);
      carrera=carr;
      curso=cur;}
      public void estudiar() { System.out.println('estudiando'); }
}
```

Ejercicio 6. Dada la siguiente clase abstracta en Java,

```
abstract class Transporte {
    protected String marca;
    protected String getMarca() { return this.marca; }
    protected void setMarca(String marc) { this.marca=marc; }
    public abstract String hacerRuta(String origen, String destino);
}
```

Crea dos nuevas clases, por ejemplo Bicicleta y NaveEspacial, que hereden de ella y que implementen el método *hacerRuta* de diferente forma cada una. Dibuja el diagrama de clases UML correspondiente a este ejercicio.

Ejercicio 7. Haz el ejercicio anterior en Ruby.

Ejercicio 8. Implementa en Java y en Ruby una jerarquía de herencia en la que estén representados los distintos tipos de instrumentos musicales: viento, percusión y cuerda. Además, los de viento pueden ser de madera o de metal. Todas las clases deben tener implementados los siguientes métodos:

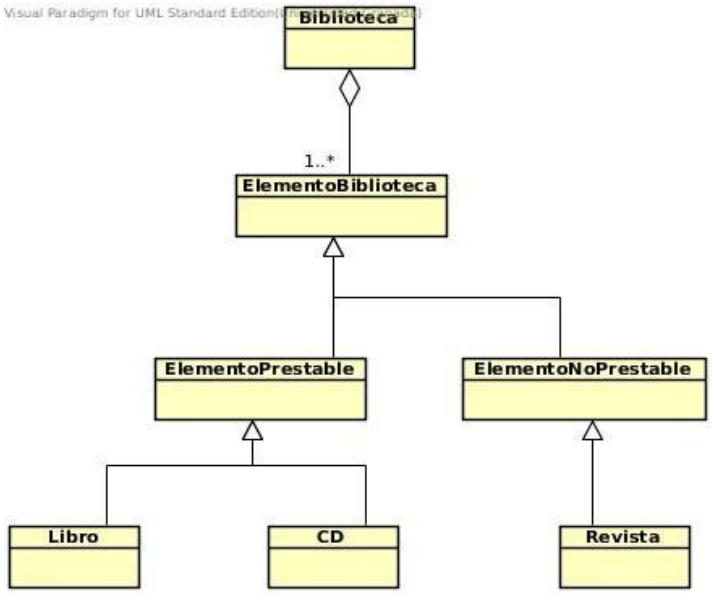
tocar(String nota)*:* muestra la nota que se le pasa (do, re, mi, fa, so, la si).
ajustar()*:* muestra "soy el instrumento fulanito y me estoy ajustando" .
queSoy()*:* muestra el tipo de instrumento que es.

Escribe un pequeño main() en el que podamos oír a la orquesta.

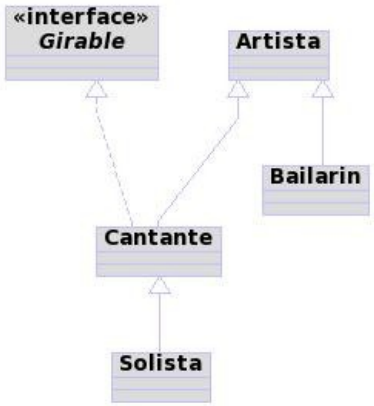
Ejercicio 9. Representa mediante un diagrama de clases las relaciones entre los distintos tipos de clases que podemos encontrarnos en Java: Clase, ClaseAbstracta, ClaseConcreta, ClaseHoja y ClaseNoHoja, siendo ClaseHoja aquella que está al final de una jerarquía de herencia (aquella que ya no tiene subclases).

Ejercicio 10. Implementa en Java una clase paramétrica a partir de la cual se puedan definir clases de grupos de personas con un líder, como por ejemplo grupos de música con un cantante, o empresas con un jefe. Implementa también alguna de esas clases a partir de la clase paramétrica definida.

Ejercicio 11. Partiendo del siguiente diagrama de clases, modifícalo añadiendo una interfaz *Prestable* implementada por la clase *ElementoPrestable* y sus subclases. Añade a este diagrama de clases los atributos y operaciones de las distintas clases y de la interfaz *Prestable*.



Ejercicio 12. Teniendo el cuenta el siguiente diagrama de clases, señala en la segunda columna de la tabla aquellas líneas de código que contienen un error de compilación por incompatibilidad de tipos (ECIT) y escribe en la tercera columna la corrección necesaria para evitarlos.



Código	ECIT	Corrección
`Girable arti=new Artista();`		
`Cantante cant1=new Cantante();`		
`Cantante sol=new Solista();`		
`Solista cant2=new Cantante();`		
`Bailarin bail= new Artista();`		

Ejercicio 13. Teniendo en cuenta:

• el diagrama de clases del ejercicio anterior,
• que se han resuelto todos los errores del ejercicio,
• que en Java la primera posición de los contenedores es la 0,
• que todos los artistas actúan, pero sólo los cantantes cantan y sólo los solistas cantan solos,

marca las líneas donde se produce un error, indicando en la segunda columna si es de compilación (C) o de ejecución (E) y en la tercera el código correcto.

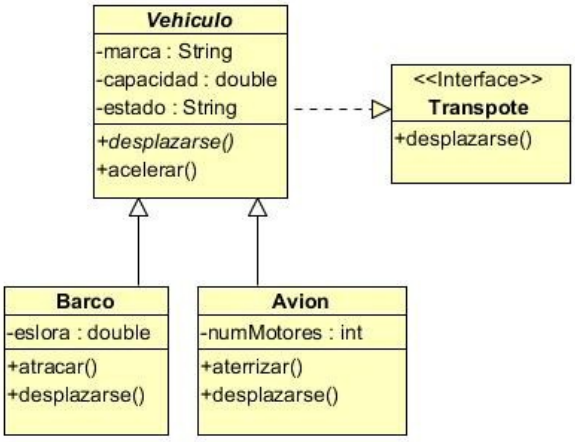
Código	Error(C/E)	Corrección
`List<Artista> lista=new ArrayList();`		

`lista.add(arti);`
`lista.add(cant1);`
`lista.add(sol);`
`lista.add(cant2);`
`lista.add(bail);`		

`lista.get(1).canta();`		
`lista.get(0).actua();`		
`(Solista)` `lista.get(3).cantaSolo();`		

Ejercicio 14. Pon tres ejemplos en Java de polimorfismo basándote en el diagrama de clases del ejercicio 8, indicando un ejemplo correcto, un ejemplo que presente errores de compilación y otro que presente un error de ejecución.

Ejercicio 15. Dado el siguiente diagrama de UML, impleméntalo en Java. Presta atención a que el nombre de la clase Vehículo y su método desplazarse() aparecen en cursiva.



Ejercicio 16. Teniendo presente el diagrama anterior, indica cuáles de los siguientes trozos de código son correctos y cuáles darían error de compilación o de ejecución. En caso de que den error, indica por qué y cómo lo solucionarías.

Código	Error y si procede corrección
`Transporte x = new Barco();` `x.atracar();`	
`Avion av = new Avion();` `av.acelerar();`	
`Vehiculo v = new Vehiculo();` `v.desplazarse();`	
`Vehiculo v2 = new Vehiculo();` `v2.acelerar();`	

`Transporte t = new Avion():`
`Barco b= new Barco();`
`t = b;`	

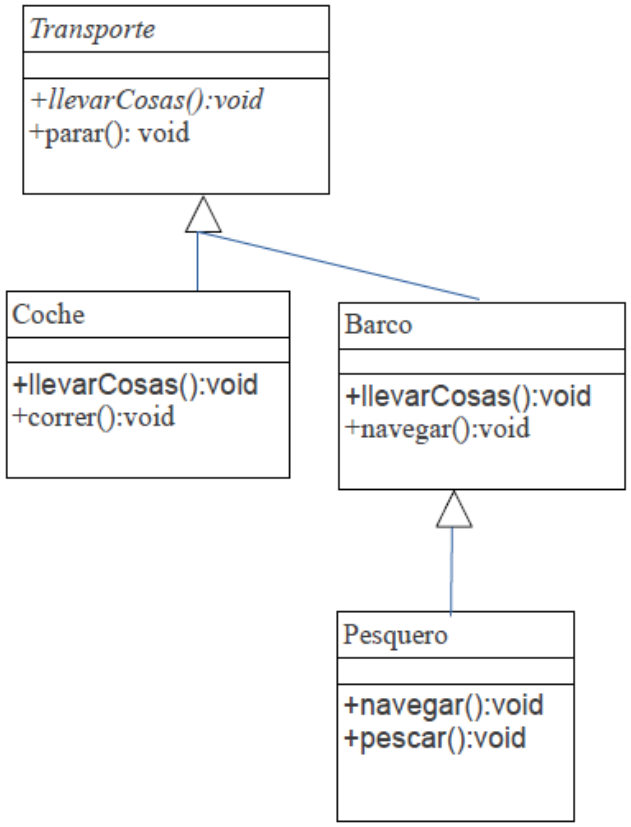
`Avion a = new Avion();` `String est = a.estado;`	
`Vehiculo v3 = new Barco();` `((Barco) v3).atracar();`	

`List<Transporte> listaTransportes = new ArrayList();`
`listaTransportes.add(new Barco());`
`listaTransportes.add(new Avion());`
`listaTransportes.add(new Barco());`
`for(Transporte tr: listaTransportes){`
`  tr.acelerar();`
`  tr.atracar();}`	

`List<Transporte> otraLista = new ArrayList();`
`otraLista.add(new Barco());`
`otraLista.add(new Avion());`
`otraLista.add(new Barco());`
`for(Transporte tr: otraLista){`
`  ((Barco) tr).acelerar();`
`  ((Barco) tr).atracar();}`	

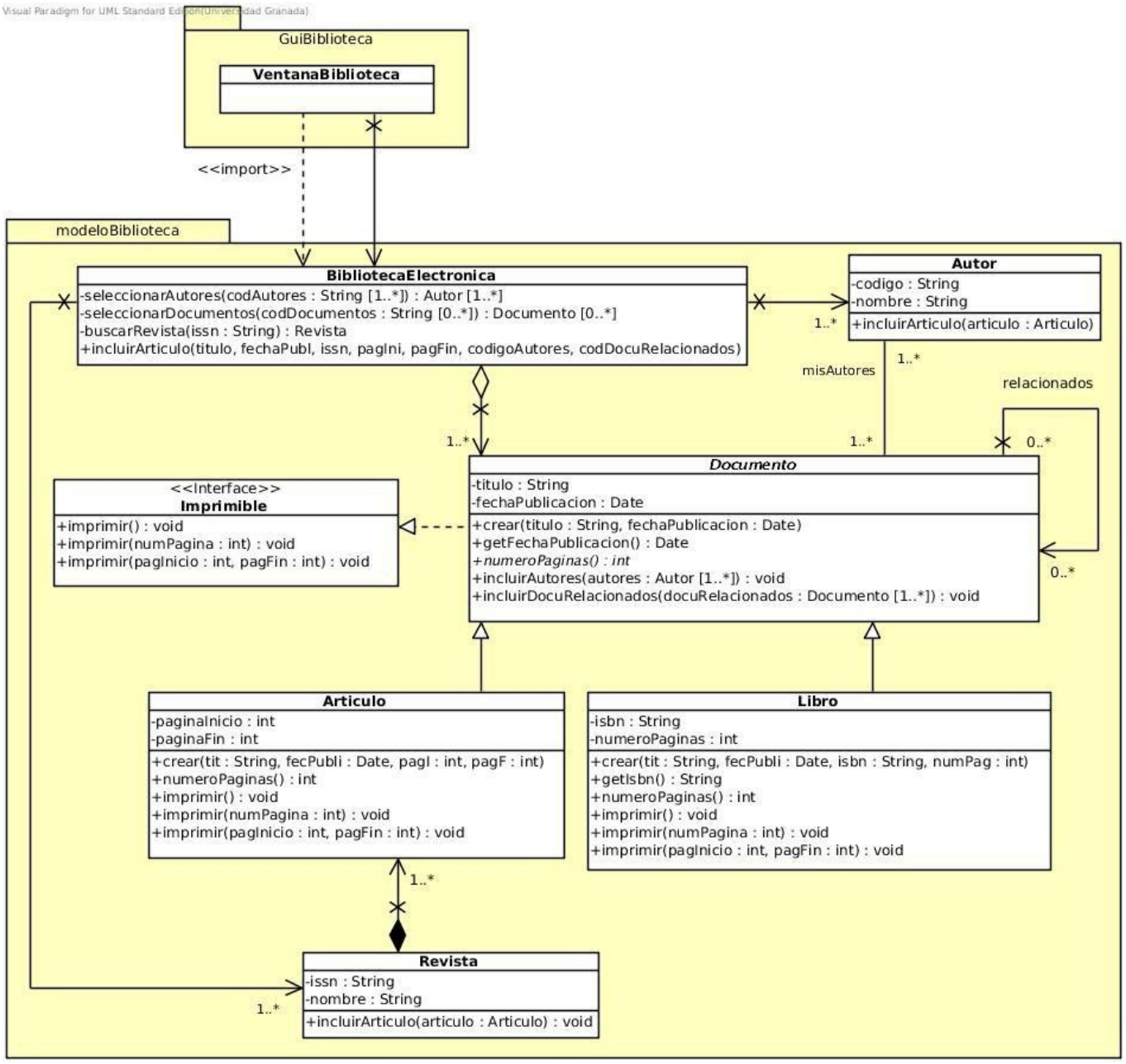
Ejercicio 17. A partir del siguiente diagrama de clases y del código proporcionado, detecta errores de compilación y de ejecución. Corrígelos para que funcione correctamente.

```
class PruebaLigadura {
  public static void main ( String args[]){

      Coche c1 = new Coche();
      c1. llevarCosas();
      c1.correr();

      Barco b = new Barco();
      b.llevarCosas();
      b.navegar();
      b.correr();
      b=c1;

      Pesquero p = new Pesquero();
      p.navegar();
      p.pescar();
      p.llevarCosas();
      p=b;
      b=p;
      b.navegar();
      b.pescar();

      ArrayList v= new ArrayList();
      v.add(c1);
      v.add(p);
      (v.get(1)).navegar();
      (v.get(1)).llevarCosas();
  }
}
```



Ejercicios de examen

Ejercicio 18. A partir de los siguientes diagramas de clases, resuelve las cuestiones que se plantea (siempre que sean de codificación hazlo en Java y en Ruby):



- Define la clase *Documento* (cabecera, atributos y cabeceras de los métodos).
- Define la clase *Articulo* (cabecera, atributos y cabeceras de los métodos).
- Implementa el constructor que se indica en la clase *Artículo*.
- Define la interfaz *Imprimible*.
- Indica los atributos que definen el estado de la clase *BibliotecaElectronica.*

- La clase *Documento* figura en cursiva, lo que indica que es abstracta. Indica los dos motivos por los que lo es.
- En este modelo, ¿puede haber artículos que no estén ligados a una revista?
- Indica sobre las relaciones del diagrama de clases con cuál de los siguientes tipos se corresponden: asociación (AS), agregación (AG), composición (C), realización (R), herencia (H), dependencia (D).
- Escribe el contenido del fichero *VentanaBiblioteca* completo (pero sin añadir nada que no aparezca en el diagrama).
- En la clase *Articulo*, indica cuáles de sus métodos están sobrecargados o redefinidos. Justifica tu respuesta.
- Corrige el código:

```
Imprimible docu = new Documento("Intemperie", fecha); // suponiendo que
fecha está inicializada
docu.imprimir();
```

- Corrige el código:

```
Documento docu = new Libro("ISBN10102030");
String codigo = docu.getIsbn();
```

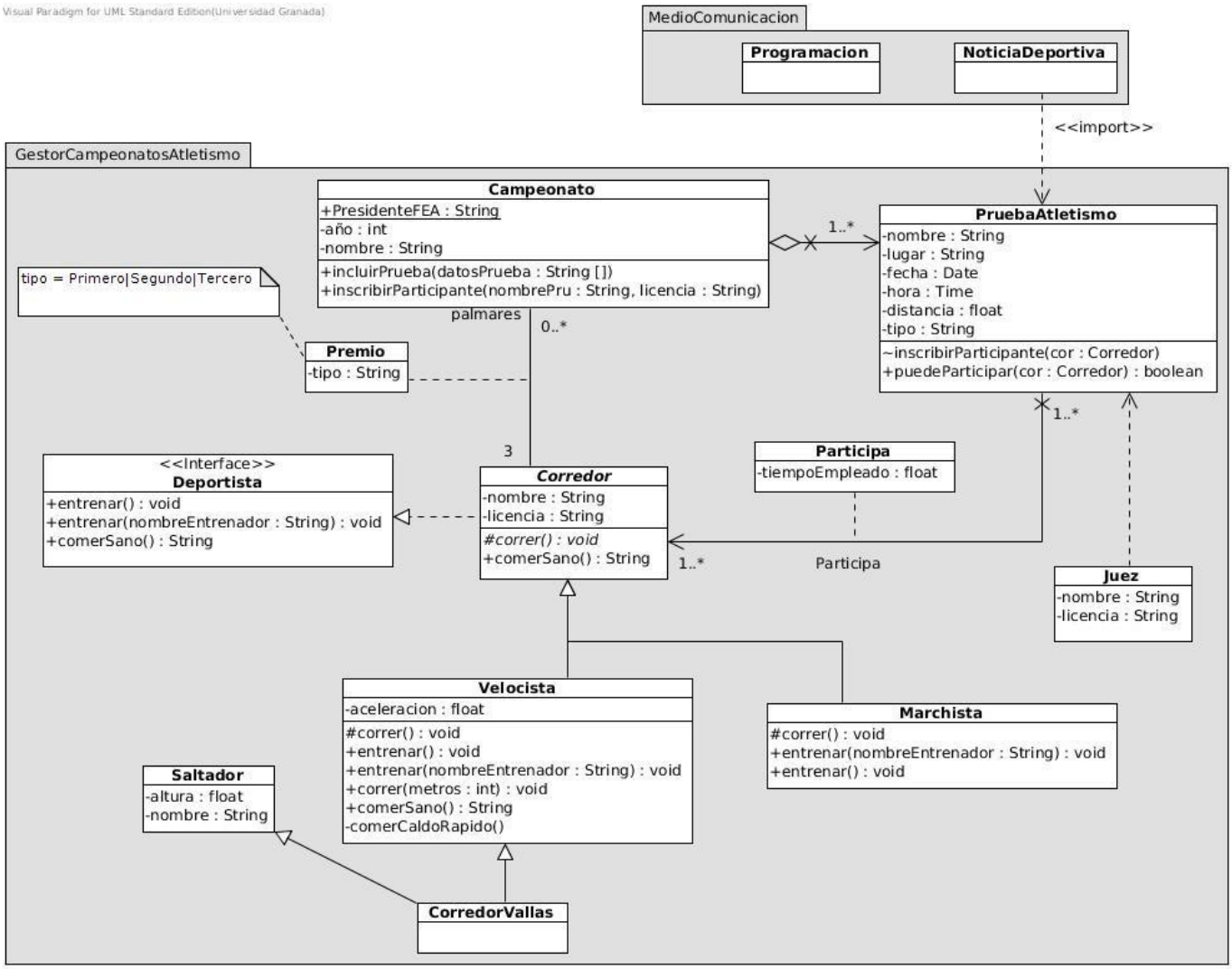
- Rellena la siguiente tabla indicando el tipo estático y dinámico de la variable *docu* en las siguientes líneas de código:

	Tipo estático	**Tipo dinámico**
`Documento docu = new Articulo(titulo, fecha, pagIni, pagFin);`		
`docu.imprimir(pagIni, pagFin);`		
`docu = new Libro(titulo, fecha, isbn, pags);`		
`docu.imprimir();`		

- Indica si hay errores de compilación o ejecución en el código anterior (suponiendo que las variables *titulo, fecha, pagIni, pagFin, isbn* y *pags* han sido declaradas e inicializadas convenientemente con anterioridad). Justifica tu respuesta.
- Rellena la siguiente tabla marcando con una "X" la situación que corresponda a cada una de las líneas del siguiente bloque de código (suponiendo que las variables *titulo, fecha, pagIni* y *pagFin* han sido declaradas e inicializadas convenientemente con anterioridad).

	Ningún error	Sólo error de compilación	Sólo error de ejecución
`Imprimible imp = new Articulo(titulo, fecha, pagIni, pagFin) ;`			
`Documento docu = imp;`			
`imp.numeroPaginas();`			
`((Libro) docu).getIsbn();`			

Ejercicio 19. Partiendo del siguiente diagrama de clases:



- Responde V (Verdarero) o F (Falso) a las siguientes cuestiones:

Desde la clase NoticiaDeportiva se puede acceder a todos los elementos públicos del paquete GestorCampeonatosAtletismo directamente	
Un CorredorVallas es un Velocista y Saltador	
El estado de un objeto de la clase Juez no viene determinado por el estado de un objeto de la clase PruebaAtletismo	
Una PruebaAtlettismo puede existir sin estar asociada a la clase Campeonato	
Un Velocista es un Corredor	
Un Deportista es un Corredor	
La clase Corredor tiene 4 métodos, 3 abstractos y 1 concreto	
La clase Marchista está mal representada en el diagrama, debe ser abstracta	
La clase CorredorVallas presenta un conflicto de nombres	
Todas las instancias de la clase Campeonato tienen una copia de la variable PresidenteFEA	

- Usando la siguiente nomenclatura: AS = Asociación, CO = Composición, AG = Agregación, DE = Dependencia, CA = Clase Asociación, RE = Realización, HE = Generalización o Herencia y HM = Herencia múltiple, etiqueta los elementos correspondientes en el propio diagrama de clases.

- Marca de los siguientes cuáles son métodos abstractos en Marchista:

`correr()`	
`comerSano()`	
`entrenar()`	
`entrenar(nombreEntrenador:String)`	

- Marca qué métodos están redefinidos y/o sobrecargados en la clase Velocista.

	redefinido	**sobrecargado**
`correr()`		
`comerSano()`		
`entrenar()`		

- Proporciona el tipo estático y dinámico de la variable que se indica en las siguientes sentencias Java.

	Variable	**Tipo Estático**	**Tipo Dinámico**
`Deportista c= new Velocista();`	`c`		
`Corredor d= new Marchista();`	`d`		
`c=new Marchista();`	`c`		
`d = c;`	`d`		

- En el siguiente código Java:
  - Corrige los posibles errores de compilación.
  - Una vez corregidos los errores de compilación, indica las líneas de código en las que habría error de ejecución.

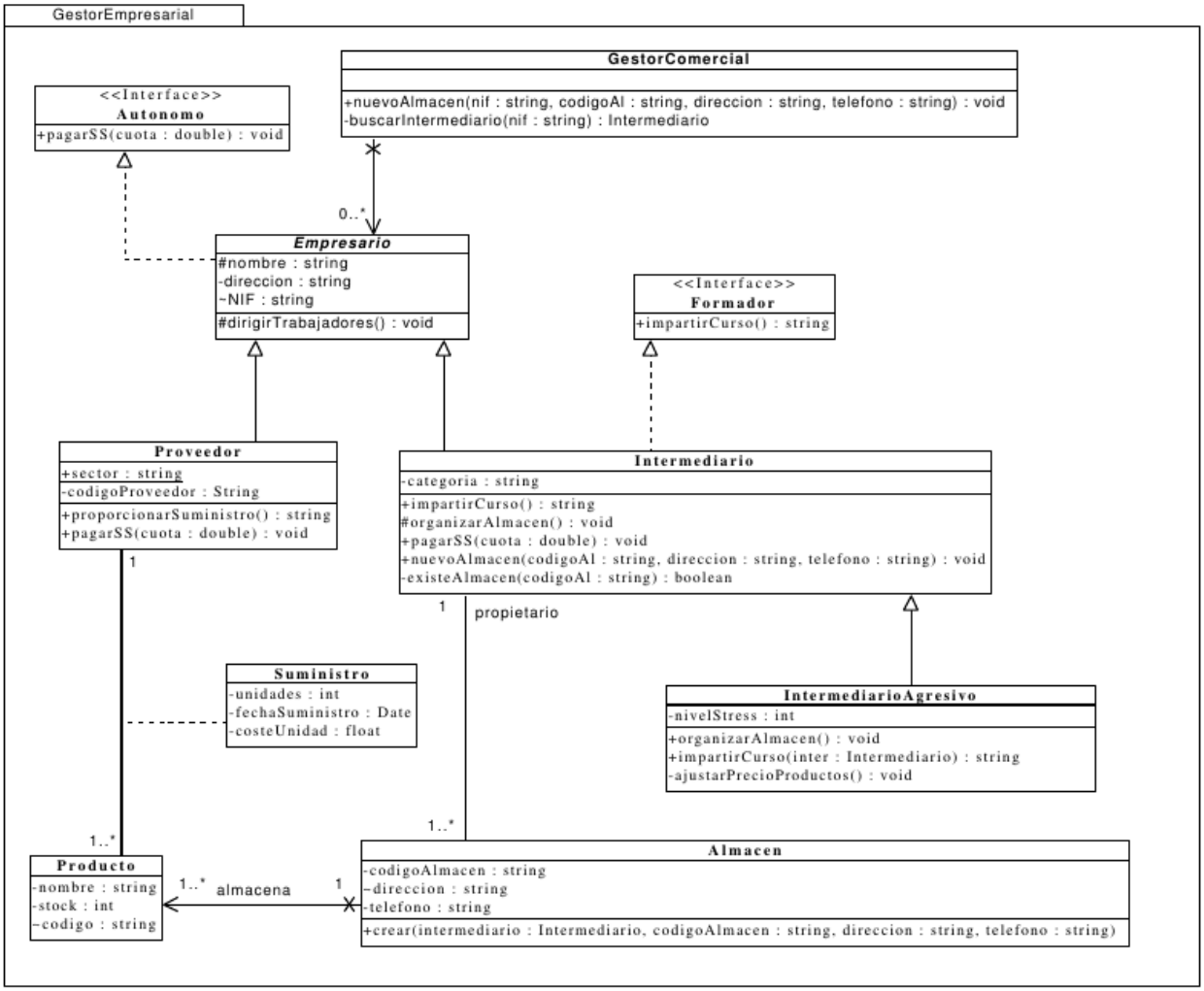
	Corrección del error en Compilación	**Error en ejecución**
`Corredor c= new Velocista();`		
`Deportista d= new Marchista();`		
`d.comerSano();`		
`Marchista m= d;`		
`d=c;`		
`d.correr(150);`		
`Velocista v = d;`		
`ArrayList<Corredor> corredores = new ArrayList();`		
`corredores.add(c);`		
`corredores.add(m);`		
`corredores.get(0).correr(10);`		
`corredores.get(1).correr(10);`		
`c= new Corredor();`		

- Completa el código de la siguiente clase parametrizada en Java, la cual representa a un club de corredores de cualquier tipo: velocistas o marchistas.

```
public class Club<......> {
       private Map<String, .....> miembros; // key = número de licencia
      private ….  lider;

       // Constructor....
       public Club(..... lider) {              }
       // Consultor de un miembro del club a partir del numero de licencia
       public ….. getMiembro(String numeroLicencia){               }
       // Incluir un nuevo miembro en el club
       public void incluirMiembro(.... miembro){            }
       // cambiar el líder
       public void setLider(..... lider) {           }
       // Obtener el lider
       public …..  getLider(){          }
}
```

- Escribe un pequeño main en el que se use la clase Club<T> como club de Velocistas

Ejercicio 20. Examen 16/17. Dado el diagrama:



Indica si las siguientes líneas de código Java producen error de compilación, de ejecución, ambos o ninguno. Supón que están en un main en una clase nueva dentro del mismo paquete. Si hay error explica cómo lo arreglarías y si no hay error, indícalo explícitamente

Considera para este ejercicio que todas las clases tienen un constructor válido que no recibe atributos.	Error de compilación	Error de ejecución
`Formador f = new Intermediario();`		
`f.pagarSS(23.4);`		
`Autonomo auto1 = f;`		
`Autonomo auto2 = (Proveedor) f;`		
`IntermediarioAgresivo emp1 = new IntermediarioAgresivo();` `emp1.ajustarPrecioProductos();`		
`Empresario emp2 = new Empresario();`		

`ArrayList<Formador> formadores = new ArrayList();`
`formadores.add(f);`
`formadores.add(emp1);`		

`formadores.get(1).impartirCurso();`		
`formadores.get(1).impartirCurso(f);`		
`((IntermediarioAgresivo)formadores.get(0)).impartirCurso(emp1);`		