Ejercicios-Tema-4.pdf

maig01

Programación y Diseño Orientado a Objetos

2º Grado en Ingeniería Informática

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación
Universidad de Granada

1.

    1. Ejemplo.new.publico_implicito_protegido

       Se imprime por pantalla el string “metodo protegido”
       Al ejecutar el método publico_implicito_protegido, este ejecuta el metodo metodo_protegido, el cual es un método de instancia que está protegido, pero se puede acceder a él desde un método público de instancia.

    2. Ejemplo.new.publico_explicito_protegido

       Se imprime por pantalla el string “metodo protegido”
       Al ejecutar el método publico_explicito_protegido, este ejecuta el metodo metodo_protegido, utilizando self.

       Un método privado nunca puede ser utilizado mediante un receptor de mensaje explícito A partir de Ruby 2.7 (diciembre 2019) sí se permite self como receptor de mensaje explícito

    3. Ejemplo.new.metodo_protegido

       Da error. No se puede acceder a un método protegido desde fuera de la clase. Hacer lo anterior es lo mismo que hacer:

       e=Ejemplo.new
       e.metodo_protegido

    4. Ejemplo.new.publico_implicito_privado

       Ejecuta un método público, por lo que es posible, el cual ejecuta un método privado, el método metodo_privado, el cual imprime por pantalla metodo privado

    5. Ejemplo.new.publico_explicito_privado

       Ocurre lo mismo que antes. Como esta versión de Ruby es a partir de 2.7 sí se puede utilizar un mensaje explícito.

    6. Ejemplo.new.metodo_privado

       Daría error, ya que se está intentando acceder a un método privado

2.

- Si lo ejecuto en la misma clase, se ejecutan todos los métodos, independientemente de su especificador de acceso.

- Si lo ejecuto en otra clase del mismo paquete, se ejecutarían los métodos públicos, los que tienen visibilidad de paquete y los protegidos, el resto darían error.
- Si lo ejecuto desde otra clase de otro paquete, sólo se ejecutaría el método público, los métodos de paquete, protegidos y privados darían error.

3.

- Al ejecutar este código en la misma clase, se ejecutarían todas las líneas del main, ya que estamos dentro de la propia clase y podemos acceder a todos los atributos
- Al ejecutarlo en una clase del mismo paquete, se imprimirían por pantalla los atributos con visibilidad pública, de paquete y protegido.
- Si lo ejecutamos en una clase de otro paquete, sólo se imprimiría por pantalla el atributo público, y el resto darían error.

4.

	Sobrecarga/Overloading	Redefinición/Overrriding
public ArrayList partesDelAbdomen();		x
public void desplazarse(Modo m);	x	
public String comunicarse(Vertebrado vertebrado);		x
public String comunicarse(Mamifero mamifero);	x	

Se redefine (sobrescribe) un método cuando una clase proporciona una implementación alternativa a la que ha heredado. La implementación heredada queda anulada En general basta con definir un método con la misma signatura (cabecera) que el método que se desea redefinir

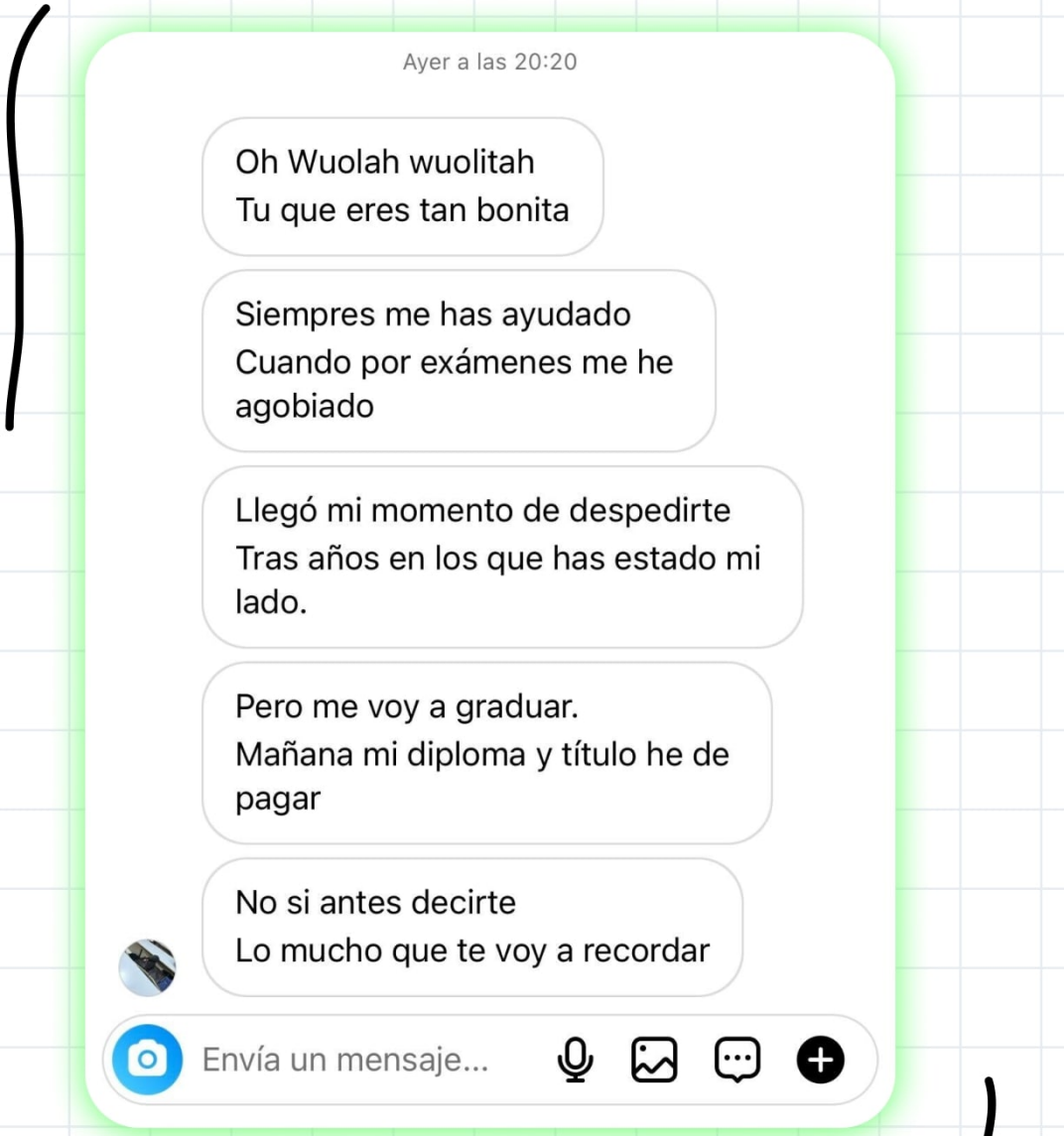
Redefinición es escribir el mismo método con la misma signatura y mismos parámetros de la clase padre. Sobrecargar es escribir el mismo método con la misma signatura pero no tiene por qué los mismo parámetros de la clase padre.

5.

```
import java.util.ArrayList;

public class Alumno-Informática extends Alumno{
```

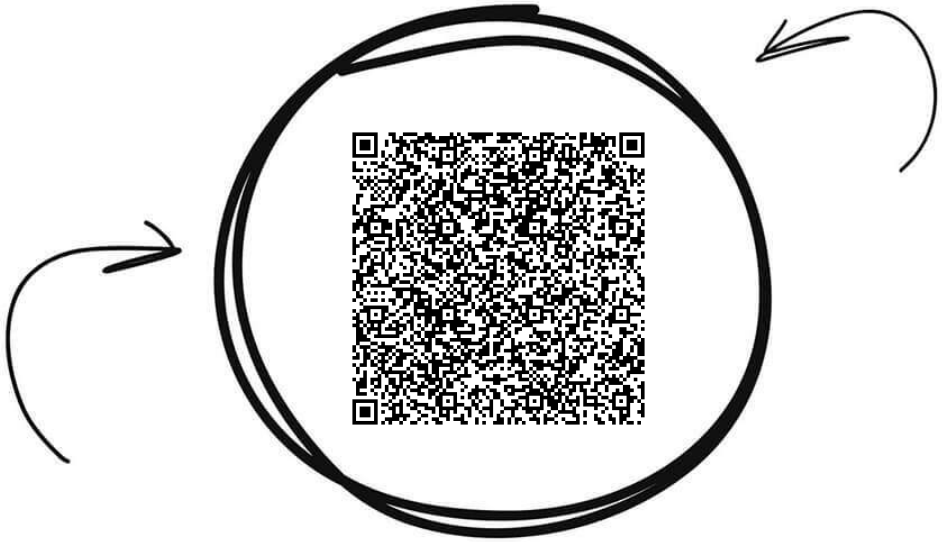
Que no te escriban poemas de amor cuando terminen la carrera 🚩🚩🚩🚩🚩🚩

(a nosotros por suerte nos pasa)



WUOLAH

Programación y Diseño Orient...



Banco de apuntes de la

WUOLAH

Comparte estos flyers en tu clase y consigue más dinero y recompensas

1. Imprime esta hoja
2. Recorta por la mitad
3. Coloca en un lugar visible para que tus compis puedan escanar y acceder a apuntes
4. Llévate dinero por cada descarga de los documentos descargados a través de tu QR

```
        private ArrayList<String> dispositivos=new ArrayList();

        public Alumno-Informática(String nom, String carr,int cur){
                super(nom,carr,cur);
        }

        public void setDispositivo(String dispositivo){
                dispositivos.add(dispositivo);
        }

        public ArrayList<String> getDispositivos(){
                return dispositivos;
        }

        public void estudiar(){
                 super.estudiar();
                System.out.println(" con "+dispositivos.get(dispositivos.size()-1));

        }
};
```

En Ruby:

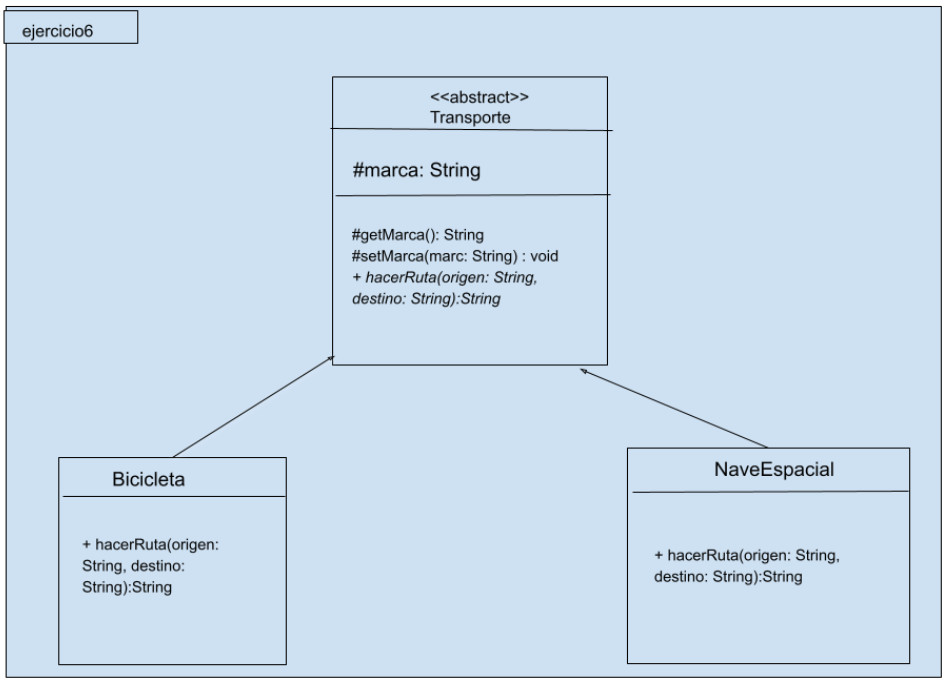
```
class Persona

         #LOs atributos son siempre privados en Ruby, por lo que esta variable la
        //vamos a poner como variable de instancia protected String nombre ;

         def initialize(nom)
                @nombre=nom
         end

         protected
         def get_nombre()
                 @nombre
         end

         protected
         def setNombre( nom)
                @nombre=nom;
         end

         def hablar
                 puts "bla bla bla"
         end
end
```

```
class Alumno < Persona

	def initialize(nom,  carr,cur)
		super(nom)
		@carrera=carr
		@curso=cur
	end

	def estudiar
		puts "estudiando"
	end
end

class AlumnoInformatica < Alumno

	def initialize( nom, carr,cur)
		super(nom,carr,cur)
		@dispositivos = Array.new
	end

	def setDispositivo( dispositivo)
		@dispositivos.push(dispositivo)
	end

	def getDispositivos
		@dispositivos
	end

	def estudiar
		super
		puts " con "+@dispositivos.at(@dispositivos.size-1)
	end
end

ai = AlumnoInformatica.new("pedor","infor",3)
dispo = "ordenador"

ai.setDispositivo(dispo);
ai.estudiar();
```

6.

```
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);
}

public class Bicicleta extends Transporte{

        @Override
        public String hacerRuta(String origen,String destino){

        }


};

public class NaveEspacial extends Transporte{

        @Override
        public String hacerRuta(String origen,String destino){

        }

};
```



7. DUDA

EN RUBY LOS ATRIBUTOS SON SIMPRE PRIVADOS, NO SE PUEDE CAMBIAR

```
class Transporte

 private_class_method :new
 attr_accessor :marca


 def initialize(marca_)
   @marca=marca_
 end

 def hacerRuta(origen,destino)
   puts "haciendo ruta con transporte"
 end
end

class Bicicleta < Transporte
 public_class_method :new
 def hacerRuta(origen,destino)
   puts "haciendo ruta con bicicleta"
 end

end



class NaveEspacial < Transporte
 public_class_method :new

 def initialize(marca_)
   super
 end

 def hacerRuta(origen, destino)
   puts "haciendo ruta con nave espacial"
 end

end

marcaaa = "hola"

nave = NaveEspacial.new(marcaaa)

nave.hacerRuta(marcaaa,marcaaa)
```

transporte_ = Transporte.new(marcaaa) → DA ERROR YA QUE AL SER TEÓRICAMENTE UNA CLASE ABSTRACTA, NO SE PUEDE CREAR UN OBJETO

DE LA CLASE TRANSPORTE PORQUE TODOS SUS MÉTODOS SON PRIVADOS, INCLUSO EL INITIALIZE

```
bici = Bicicleta.new(marcaaa)
bici.hacerRuta(marcaaa,marcaaa)
```

8.

```
abstract class Instrumento {
   public String tocar(String nota){
         return nota;
}

public void ajustar(){
    System.out.println(queSoy()+"y me estoy ajustando");
}
public abstract String queSoy();

}

public class Viento extends Instrumento {

   @Override
   public String queSoy() {
      return "Soy en instrumento Viento";
   }

}

public class Percusión extends Instrumento {


   @Override
   public String queSoy() {
      return "Soy el instrumento percusión";
   }

}

public class Cuerda extends Instrumento{


   @Override
   public String queSoy() {
      return "Soy el instrumento Cuerda";
   }
```

```
}

public class VientoMadera extends Viento{
   @Override
   public String queSoy(){
         return "Soy el instrumento viento Madera";
   }

}

public class VientoMetal extends Viento{
   @Override
public String queSoy(){
         return "Soy el instrumento viento metal";
}

}

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

        Viento i1 = new Viento();
        i1.ajustar();
        VientoMetal i2 = new VientoMetal();
        i2.ajustar();

   }
}
```

9.
10.

```
public class Grupo<T>{ //T puede ser una empresa, un grupo de musica, etc
         private String lider;
}
```

11.
12.

Código	ECIT	Corrección
Girable arti=new Artista();	x	Girable arti=(Girable) new Artista(); ESTA CORRECCIÓN DA UN ERROR DE EJECUCIÓN

Cantante cant1=new Cantante();		
Cantante sol=new Solista();		
Solista cant2=new Cantante();	x	Solista cant2=(Solista) new Cantante(); ESTA CORRECCIÓN DA UN ERROR DE EJECUCIÓN
Bailarin bail= new Artista();	x	Bailarin bail= (Bailarin) new Artista(); ESTA CORRECCIÓN DA UN ERROR DE EJECUCIÓN

13.

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

lista.add(arti); x
lista.add(cant1);
lista.add(sol);
lista.add(cant2);
lista.add(bail);
	compilación da error porque arti es de tipo girable, y girable es una interfaz	lista.add((Artista) arti)

lista.get(1).canta();	compilación	((Cantante)lista.get(1)).canta()
lista.get(0).actua();		((Artista)lista.get(0)).actua()
(Solista) lista.get(3).cantaSolo();	compilación	((Solista)lista.get(3)).cantasolo();

14.

- Correcto

```
VientoMetal i3 = (VientoMetal) new Viento();
 ArrayList<Instrumento> i = new ArrayList<>();

  i.add(new Cuerda());
  i.add(new Viento());
  i.add(new VientoMadera());
```

- Error de compilación

```
i.get(0).TocarCuerdas();
```

- Error de ejecución

```
VientoMetal metalico = ((VientoMetal)i.get(1));
metalico.TocaMetalico(); //error ejecucion
Cuerda cuerda = ((cuerda)i.get(0));
cuerda.tocarCuerda();//OK
```

15.

```
interface Transporte{
        public void desplazarse();
}

abstract class  Vehiculo implements Transporte{
        private String marca;
        private double capacidad;
        private String estado;

        public abstract void desplazarse();
        public void acelerar(){}
};

public class Barco extends Vehiculo{
        private double eslora;
        public void atracar(){}

        @Overrride
        public void desplazarse(){}
};

public class Avion extends Vehiculo{
        public int numMotores;

        public void aterrizar(){}
        @Overrride
        public void desplazarse(){}
};
```

16.

DIFERENCIAR ENTRE ERROR DE COMPILACIÓN Y ERROR DE EJECUCIÓN

Código	Error y si procede corrección
Transporte x = new Barco(); x.atracar(); ERROR	((Barco) x).atracar();
Avion av = new Avion();	

av.acelerar();	
Vehiculo v = new Vehiculo(); ERROR v.desplazarse();	Vehiculo v = (Vehiculo) new Barco(); v.desplazarse();
Vehiculo v2 = new Vehiculo(); v2.acelerar();	Vehiculo v2 = (Vehiculo) new Barco(); v2.acelerar();
Transporte t = new Avion(); Barco b= new Barco(); t = b;	No da ningun error ni al compilar ni al ejecutarlo
Avion a = new Avion(); String est = a.estado;	Da error ya que estado es una variable que avion hereda de vehículo y es una variable privada. Se podría acceder a ella mediante un consultor publico o protegido, pero NUNCA privado.
Vehiculo v3 = new Barco(); ((Barco) v3).atracar();	NO da ni error de compilación ni error de ejecución

List listaTransportes = new ArrayList();
listaTransportes.add(new Barco());
listaTransportes.add(new Avion());
listaTransportes.add(new Barco());
for(Transporte tr: listaTransportes){
tr.acelerar(); tr.atracar();}
	Transporte tr = (Transporte)
	listaTransportes.get(i);
	((Vehiculo)tr).acelerar();
	((Barco) tr).atracar(); ESTO DA UN ERROR DE EJECUCION, NO SE COMO SE PONDRIA

List 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();}
	((Barco) tr).acelerar(); ((Barco)
	tr).atracar() dan errores de ejecucion y no se como se solucionan

17.

```
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(); //ERROR DE COMPILACION

        b=c1; //ERROR DE COMPILACION

        Pesquero p = new Pesquero();
        p.navegar();
        p.pescar();
        p.llevarCosas();
        p=b; //ERROR DE COMPILACION
         p=(Pesquero) b; //CORRECCION, DA ERROR DE EJECUCION
        b=p;
        b.navegar();
        b.pescar(); //ERROR DE COMPILACION
         ((Pesquero) b).pescar(); //CORRECCION


        ArrayList v= new ArrayList();
        v.add(c1);
        v.add(p);
        (v.get(1)).navegar(); //ERROR DE COMPILACION
        ((Pesquero) v.get(1)).navegar(); //CORRECCION
        (v.get(1)).llevarCosas(); //ERROR DE COMPILACION
        ((Pesquero) v.get(1)).llevarCosas(); //CORRECCION
    }
}
```

18.

- Define la clase Documento (cabecera, atributos y cabeceras de los métodos).

```
package modeloBiblioteca;

import java.util.ArrayList;
import java.util.Date;

abstract class Documento implements Imprimible {

   private String titulo;
   private Date fechaPublicacion;
   private ArrayList<Autor> misAutores;
   private ArrayList<Documento> relacionados;

   //EN EL DIAGRAMA PONE QUE EL METODO SE LLAMA CREAR, PERO
PONEMOS COMO QUE ES UN CONSTRUCTOR?
   public Documento(String titulo, Date fechaPublicacion){

   }
```

```
    public Date getFechaPublicacion(){
        return null;
    }

    public abstract int numeroPaginas();

    public void incluirAutores(ArrayList<Autor> autores){}

    public void incluirDocuRelacionados(ArrayList<Documento>
docuRelacionados){}
}
```

- Define la clase Articulo (cabecera, atributos y cabeceras de los métodos).

```
package modeloBiblioteca;

import java.util.Date;


public class Articulo extends Documento{

    private int paginaInicio;
    private int paginaFin;

    public Articulo(String tit, Date fecPubli,int pagI,int pagF) {
        super(tit, fecPubli);
        paginaInicio=pagI;
        paginaFin=pagF;
    }

    @Override
    public int numeroPaginas() {
        return 0;
        }

    @Override
    public void imprimir() {

    }

    @Override
    public void imprimir(int numPagina) {
    }

    @Override
    public void imprimir(int paginicio, int pagFin) {
     }
```

```
}
```

- Implementa el constructor que se indica en la clase Artículo.

```
public Articulo(String tit, Date fecPubli,int pagI,int pagF) {
      super(tit, fecPubli);
      paginaInicio=pagI;
      paginaFin=pagF;
  }
```

- Define la interfaz Imprimible

```
package modeloBiblioteca;


interface Imprimible {
   public void imprimir();
   public void imprimir(int numPagina);
   public void imprimir(int paginicio,int pagFin);
}
```

- Indica los atributos que definen el estado de la clase BibliotecaElectronica.

```
   private ArrayList<Autor> autores;
   private ArrayList<Revista> revistas;
   private ArrayList<Documento> documentos;
```

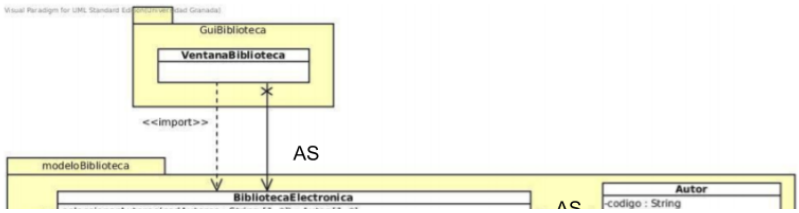
- La clase Documento figura en cursiva, lo que indica que es abstracta. Indica los dos motivos por los que lo es.

  Representa de manera genérica a otras entidades que sí concretan el funcionamiento desconocido. Definen dos tipos de datos comunes para todos los tipos de documentos y obliga a las clases hijas a implementar el metodo numPaginas y los metodos de la interface que implementa Documento, ya que en esta clase no se han implementado, y tienen que ser implementados por las clases hijas

- En este modelo, ¿puede haber artículos que no estén ligados a una revista?

  No, ya que tienen una relación de composición, donde los artículos no tienen sentido sin la 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).



En la flecha discontinua de imprimible pondríamos realización

- Escribe el contenido del fichero VentanaBiblioteca completo (pero sin añadir nada que no aparezca en el diagrama).

```
package GuiBiblioteca;

import modeloBiblioteca.BibliotecaElectronica;

public class VentanaBiblioteca {
   private BibliotecaElectronica biblioteca;
}
```

- En la clase Articulo, indica cuáles de sus métodos están sobrecargados o redefinidos. Justifica tu respuesta.

```
package modeloBiblioteca;

import java.util.Date;


public class Articulo extends Documento{

   private int paginaInicio;
   private int paginaFin;

   public Articulo(String tit, Date fecPubli,int pagI,int pagF) {
      super(tit, fecPubli);
      paginaInicio=pagI;
      paginaFin=pagF;
   }

   @Override
   public int numeroPaginas() {
      return 0;
      }

   @Override
   public void imprimir() {

   }

   @Override
   public void imprimir(int numPagina) {
   }
```

```
    @Override
    public void imprimir(int paginicio, int pagFin) {
     }

}
```

- 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();

  Documento docu = new Libro("hola",fecha,"ISBN10102030",4);
  String codigo = ((Libro)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);	Documento	Articulo
docu.imprimir(pagIni, pagFin);	Documento	Documento
docu = new Libro(titulo, fecha, isbn, pags);	Documento	Libro
docu.imprimir();		

Tipo estático: tipo (clase) del que se declara la variable
Tipo dinámico: clase al que pertenece el objeto referenciado en un momento determinado por una variable

- 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.

  - Documento docu = new Articulo(titulo, fecha, pagIni, pagFin); NO
  - docu.imprimir(pagIni, pagFin); NO, ya que es un objeto de Articulo, EJECUTA LA FUNCION IMPRIMIR DE LA CLASE ARTICULO
  - docu = new Libro(titulo, fecha, isbn, pags); NO DA ERROR

- docu.imprimir(); NO DA ERROR, Y EJECUTA LA FUNCION IMPRIMIR() DE LIBRO, YA QUE ANTES HEMOS CAMBIADO DOCU Y AHORA ES UN OBJETO DE LA CLASE LIBRO

- 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	Error de compilación	Error de ejecución
Imprimible imp = new Articulo(titulo, fecha, pagIni, pagFin) ;	x		
Documento docu = imp;		x Documento docu = (Document o) imp;	
imp.numeroPaginas();		x	
((Libro) docu).getIsbn();			x NO SE POR QUE DA ERROR DE EJECUCIO N da error porque teoricament e docu3 es un articulo, y no se puede pasar de articulo a libro

En este apartado notamos una diferencia respecto al apartado anterior.

Imprimible imp = new Articulo(titulo, fecha, pagIni, pagFin) ;

Al crear este objeto, tiene tipo estático Imprimible y tipo dinámico Articulo. Desde imp sólo podemos acceder a los métodos de Imprimible que están implementados en Articulo, pero no a los demás

Documento docu = new Articulo(titulo, fecha, pagIni, pagFin);

Desde docu podemos acceder a todos los elementos de Articulo, porque es una clase hija

19.

- 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	V Hay que importar el paquete
Un CorredorVallas es un Velocista y Saltador	V
El estado de un objeto de la clase Juez no viene determinado por el estado de un objeto de la clase PruebaAtletismo	V
Una PruebaAtlettismo puede existir sin estar asociada a la clase Campeonato	V
Un Velocista es un Corredor	V
Un Deportista es un Corredor	F
La clase Corredor tiene 4 métodos, 3 abstractos y 1 concreto	F
La clase Marchista está mal representada en el diagrama, debe ser abstracta	F
La clase CorredorVallas presenta un conflicto de nombres	V
Todas las instancias de la clase Campeonato tienen una copia de la variable PresidenteFEA	F, es una variable static

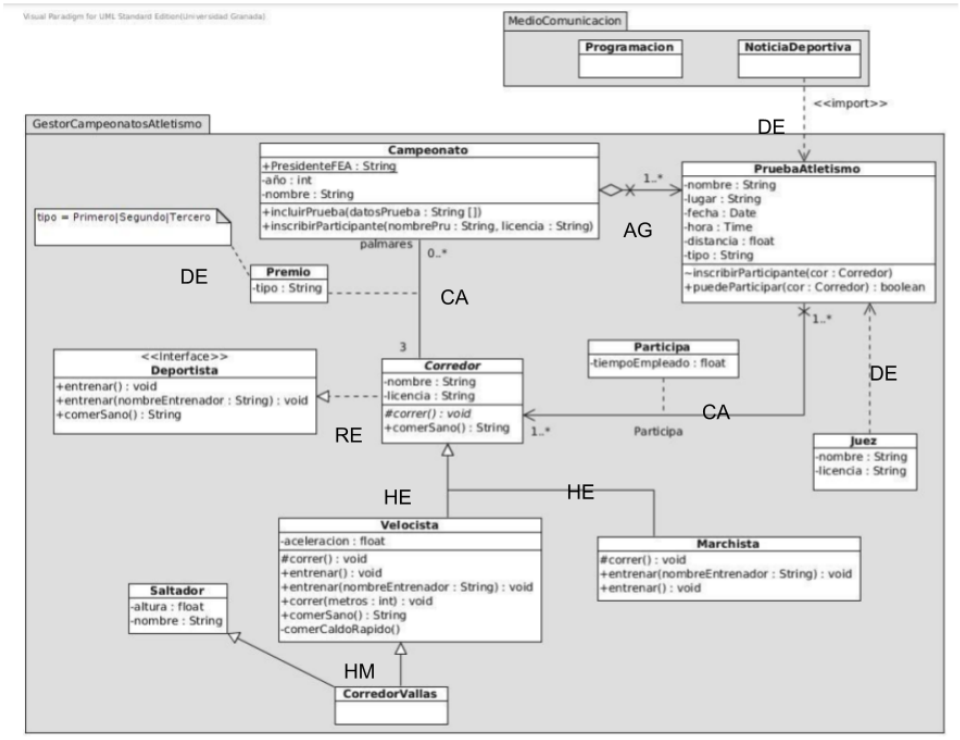
Las clases que heredan de una clase abstracta, si no implementan algún método también serán clases abstractas

Corredor es una clase abstracta que implementa deportista. En el dibujo de corredor sólo aparece que implemente el método comerSano() de Deportisa, por lo que el resto de métodos de Deportista tienen que estar implementados en sus clases hijas

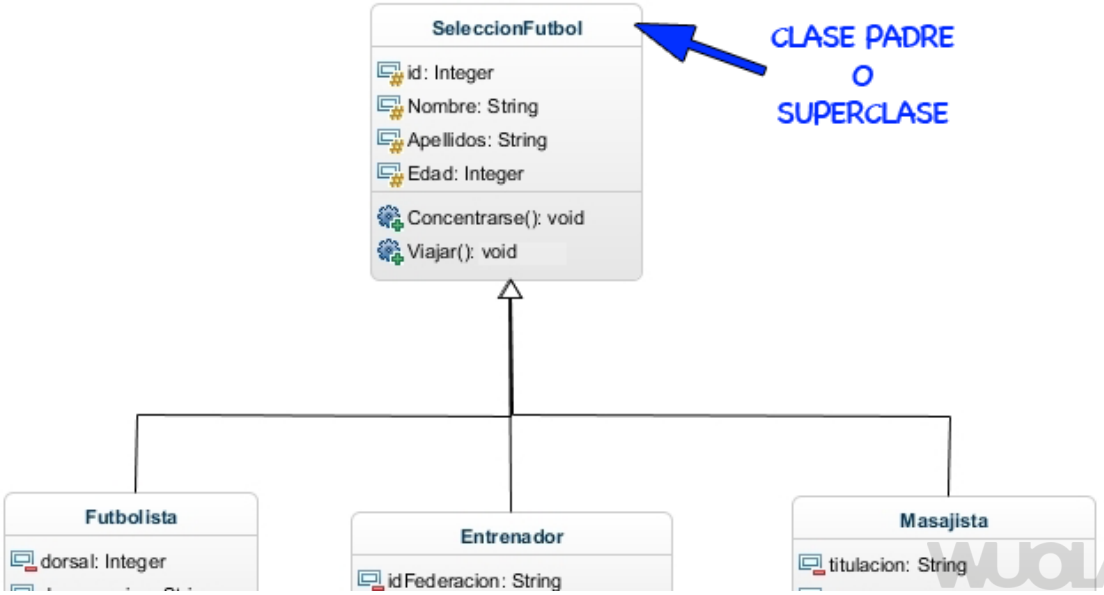
Una clase abstracta puede indicar que realiza una interfaz sin implementar alguno de sus métodos. I Fuerza a hacerlo a sus descendientes no abstractos

- 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.

Realización es cuando una clase interpreta una interfaz



En la herencia, cuando hay varias clases hijas, no hace falta que todas tengan la punta de la flecha, puede representarse así también.



si lees esto me debes un besito

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

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

Métodos abstractos son los de las clases abstractas y los de las clases interface (CREO)

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

	Redefinido	Sobrecargado
correr()	x	
comerSano()	x	
entrenar()	x	

- 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	Deportista	Velocista
Corredor d= new Marchista();	d	Corredor	Marchista
c=new Marchista();	c	Deportista	Marchista
d=c	d	Corredor	Deportista

- 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;	Marchista m = (Marchista) d;	
d=c;		
d.correr(150);	((Corredor) d).correr();	
Velocista v = d;	Velocista v = (Velocista) d;	
ArrayList corredores = new ArrayList();		
corredores.add(c);		
corredores.add(m);		
corredores.get(0).correr(10);	((Corredor) corredores.get(0)).correr();	
corredores.get(1).correr(10);	((Corredor) corredores.get(1)).correr();	
c= new Corredor();	x no se como se corrige	

- 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<T extends Corredor> {

        private Map miembros; // key = número de licencia
        private T lider;

        // Constructor....
        public Club(T lider) { }

        // Consultor de un miembro del club a partir del numero de licencia
        public T getMiembro(String numeroLicencia){ }
```

```
    // Incluir un nuevo miembro en el club
    public void incluirMiembro(T miembro){
    }

    // cambiar el líder
    public void setLider(T lider) { }

    // Obtener el lider
    public T getLider(){ }
}
```

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

```
    Velocista v2;
v=new Velocista();
Club<Velocista> club;

club = new Club<>(v);

Velocista v3;
v3 = new Velocista();

club.incluirMiembro(v3);
```

20. 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);	((Intermediario) f).pagarSS(23.4);	
Autonomo auto1 = f;	Autonomo auto1 = (Autonomo) f;	
Autonomo auto2 = (Proveedor) f;		ERROR DE EJECUCIÓN, ya que f es de Formador, y proveedor no esta relacionado

		con Formador
IntermediarioAgresivo emp1 = new IntermediarioAgresivo(); emp1.ajustarPrecioProductos();	Error de compilación ya que ajustarPrecioProductos es un método privado, y para arreglarlo podríamos ponerle visibilidad de paquete, protegida,	
Empresario emp2 = new Empresario();	Error de compilación Empresario emp2; NO DA ERROR	
ArrayList formadores = new ArrayList(); formadores.add(f); formadores.add(emp1);		
formadores.get(1).impartirCurso();	((Intermediario) formadores.get(1)).impartirCurso()	
formadores.get(1).impartirCurso(f);	((IntermediarioAgresivo) formadores.get(1)).impartirCurso((Intermediario)f);	
((IntermediarioAgresivo)formadores.get(0)).impartirCurso(emp1);		Da error de ejecución, ya que no se puede pasar de un Formador a un intermediario agresivo

21.
22.
23.
24.