Programación y Diseño Orientado a Objetos.
Grado en Ingeniería Informática
Examen de teoría. Curso 2021-2022.
17 de enero de 2022. Examen tipo 3

Tiempo: 45 minutos

[redacted]

1.- (2 puntos) Dado el siguiente código

```
abstract class Profesor {
   private String nombre;

   Profesor(String nombre) {
      this.nombre=nombre;
   }
   abstract void impartirDocencia();
}

interface Investigador {
   public void investigar();
}

class ProfesorSecundaria extends Profesor {

   ProfesorSecundaria(String nombre) {
      super(nombre);
   }

   @Override
   void impartirDocencia() {
      System.out.println("Dando clase en un instituto");
   }

   void vigilarRecreo() {
      System.out.println("Vigilando el recreo en un instituto");
   }
}

class ProfesorUniversidad extends Profesor implements Investigador {

   ProfesorUniversidad(String nombre) {
      super(nombre);
   }

   @Override
   void impartirDocencia() {
      System.out.println("Dando clase en una facultad");
   }
```

```
    @Override
    public void investigar() {
        System.out.println("Investigando en una facultad");
    }

}

class ProfesorUniversidadDoctor extends ProfesorUniversidad {

    ProfesorUniversidadDoctor(String nombre) {
        super(nombre);
    }

    @Override
    public void investigar() {
        super.investigar();
        System.out.println("Como doctor");
    }
}
```

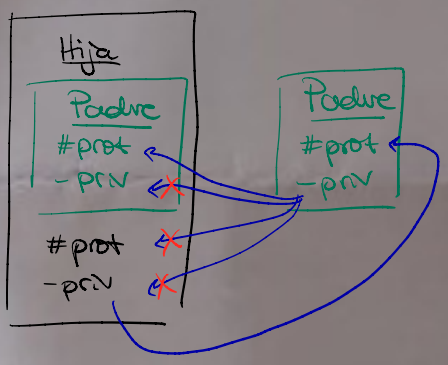
Indicar los errores relacionados con compatibilidad de tipos presentes en el siguiente programa principal. Se deberá indicar el número de línea, si se produce en tiempo de compilación o ejecución y dar una muy breve descripción de la causa.

```
1.  public class Enero22 {
2.      public static void main(String[] args) {
3.          ((ProfesorUniversidad) new ProfesorUniversidadDoctor("Manolo")).investigar();
4.          Profesor prof1=new ProfesorUniversidad("Pepe");
5.          prof1.investigar();
6.          Investigador inv1=prof1;
7.          ProfesorSecundaria secu1=(ProfesorSecundaria) (Profesor) new ProfesorUniversidad("Juan");
8.      }
9.  }
```

Línea	Compilación/ejecución	Descripción
5	Compilación	Para el compilador prof1 es un objeto de Profesor, para solucionarlo había que hacer un casting: ((ProfesorUniversidad) prof1).investigar();
6	Compilación	No se pueden crear objetos de una interfaz.
7	Ejecución	No se puede aplicar polimorfismo de una clase hermana

2.-(2 puntos) Dado el siguiente código

```
1.  module Primero
2.  class Padre
3.
4.    def metodoPublico(p)
5.      metodoPrivado          //ok
6.    p.metodoPrivado          // no
7.      metodoProtegido        //ok
8.      p.metodoProtegido      // ok
9.    end
10.
11.   private
12.   def metodoPrivado
13.   end
14.
15.   protected
16.   def metodoProtegido
17.   end
18. end
19.
20. end
21. - - - - - - -
22. module Segundo
23.   class Hija<Primero::Padre
24.
25.     def metodoPublico(p)
26.       metodoPrivado          //ok
27.     p.metodoPrivado          // no
28.       metodoProtegido        //ok
29.       p.metodoProtegido      // ok
30.     end
31.
32.   end
33. end
34. - - - - - -
35. Primero::Padre.new.metodoPublico(Segundo::Hija.new)
36. Segundo::Hija.new.metodoPublico(Primero::Padre.new)
```



Empezando desde las líneas 35 y 36 indicar en las líneas en las que se produce algún error relacionado con la accesibilidad (visibilidad) de métodos, asumiendo que cada error es ignorado y que la ejecución puede continuar.

Respecto a la línea 35:
→ se produciría un error en la línea 6 debido a que el método "metodoPrivado" solo es accesible por objetos de la clase "Padre".

Respecto a la línea 36:
→ se produciría un error en las líneas 26 y 27 ya que como he dicho antes el método "metodoPrivado" solo es accesible para la clase "Padre".

3.-(2 puntos) Dado el siguiente código

```
interface Doblable {
   void doblar();
}

class Mapa implements Doblable {
   @Override
   public void doblar(){
      //....
   };
}

class Alfombra implements Doblable {
   @Override
   public void doblar(){
      //....
   };
}

class Kebab implements Doblable {
   @Override
   public void doblar(){
      //....
   };
}
```

Este ejercicio se centra en la operación para enrollar mapas, alfombras y kebabs. Sobre esta operación se proporciona la siguiente información:

- La operación de enrollado, cuyo código no se proporciona, se debe asumir que tiene cierta complejidad (unas 100 líneas de código) y que se basa siempre en el uso progresivo e iterativo del método *doblar* del elemento que se desea enrollar.
- El enrollado de un mapa, de un kebap y de una alfombra se hace exactamente de la misma forma (usando, como ya se ha comentado, el método *doblar* de estos elementos).
- Se desea crear las clases EnrolladorDeMapas, EnrolladorDeAlfombras y EnrolladorDeKebaps, de forma que dispongan todas de un método de instancia llamado *enrollar* que se aplique a un parámetro del tipo adecuado en cada caso (mapa, alfombra y kebab respectivamente).
- Es importante que un EnrolladorDeMapas solo permita enrollar mapas, el EnrolladorDeAlfombras solo permita enrollar alfombras y el EnrolladorDeKebaps solo permita enrollar kebaps. De esta comprobación debe encargarse el compilador.
- Es probable que en el futuro aparezcan más elementos doblables y que también se tengan que crear clases que permitan enrollarlos. → Parametrizable

Proporcionar una implementación, utilizando el lenguaje de programación Java, para las tres clases indicadas. Del método *enrollar*, solo se deberá proporcionar la cabecera completa (incluyendo el parámetro).