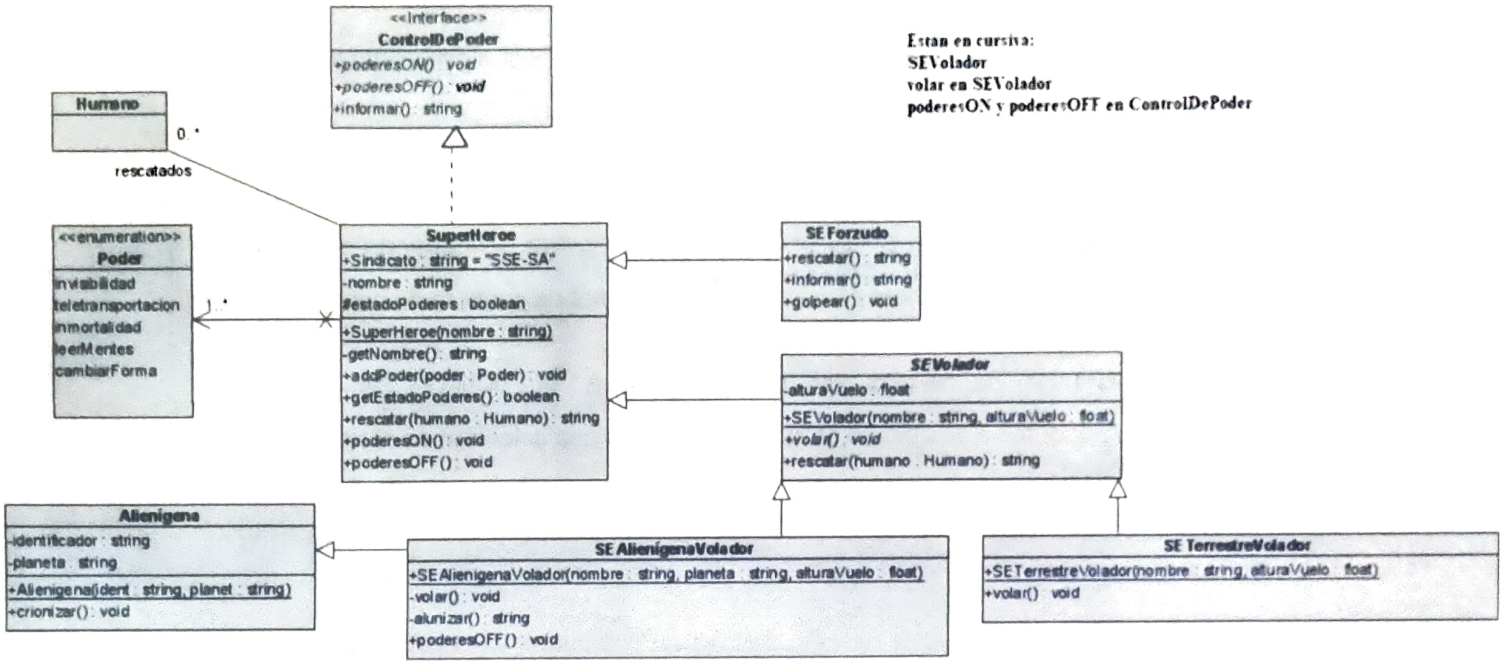

---

1. Indica verdadero o falso:

- La relación entre ControlDePoder y SuperHeroe es de dependencia

  Falso, es una relación de realización, ya que se está implementando una interfaz.

- SEVolador es una interfaz porque está en cursiva, aunque no lleve <<interface>>, es otra forma de representarla.

  Falsa, al estar en cursiva y tener al menos un método abstracto se trata de una clase abstracta.

- En el enumerado Poder hay un atributo de la clase SuperHeroe

  Falso, ya que solo se puede navegar desde SuperHeroe hasta Poder, y no al contrario.

- En la interface ControlDePoder podría añadirse una constante, por ejemplo, DuracionPoderes, siempre que sea protected

  Falso, las variables de las constantes solo pueden ser públicas y/o static y/o final.

- El método volar está redefinido en las clases que heredan de SEVolador

  Verdadera, está redefinido en ambas subclases de SEVolador, aunque tengan distinta visibilidad.

- El método informar está sobrecargado en SEForzudo

  Falso, está redefinido a partir del método informar de la interfaz ControlDePoder

- Un objeto de la clase SEForzudo no puede consultar su atributo nombre en Java

  Verdadero, para poder hacer la consulta será necesario hacer super de nombre y el consultor getNombre()

- A los objetos de la clase SEVolador se les puede enviar el mensaje informar

  Falso, no se puede instanciar una clase abstracta

- La clase SEAlienigenaVolador hereda todos los atributos de SEVolador excepto los privados

  Verdadero

- Si SETerrestreVolador fuera una clase abstracta necesitaría añadir al menos un método abstracto

  Verdadero

- Si en SuperHeroe estuviera el atributo identificador, habría conflicto de nombres en SEAlienigenaVolador

  Falso, para que hubiese colisión tendría que existir el atbo. identificador en SuperHeroe y que se declarara sin inicializar en SEAlienigenaVolador.

- Si en la clase SEVolador estuviera el método crionizar(), habría conflicto de nombres en la clase SEAlienigenaVolador

  Verdadero, heredaría dos métodos con el mismo nombre de distintas ramas

- La clase SEVolador tiene un atributo de referencia que es una colección de objetos del enumerado Poder

  Verdadero

- En el siguiente código:

```
SuperHeroe batman= new SETerrestreVolador("Batman", "Tierra", 1.2);
batman.rescatar();
```

hay ligadura dinámica en el método rescatar que se debe ejecutar se decide en tiempo de ejecución dependiendo de la clase de la variable batman

Falso, hay un error al llamar al constructor de SETerrestreVolador, solo se le deben pasar dos parámetros ⟶ sobra "Tierra"

- Hay error en este código Ruby en el método alunizar de SEAlienigenaVolador (donde otro es un objeto conocido de la clase SuperHeroe y nombre es el consultor del atributo con el mismo nombre)

```
puts "el nombre de mi amigo es:  " + otro. nombre + " y rescatamos humanos en peligro"
```

Verdadero, ya que según el diagrama el consultor de nombre es privado en la clase SuperHeroe

- Hay error en este código Java en el método rescatar de SuperHeroe

```
return "me llamo " + nombre + " y rescato humanos en peligro"
```

Falso, se puede acceder sin problemas al atributo nombre

- En el siguiente código en Java, el tipo estático de la variable superman es SEAlienigenaVolador y su tipo dinámico SEAlienigenaVolador

```
SEVolador superman= new SEAlienigenaVolador("superman", "Krypton", 5.8);
```

Falso, su tipo dinámico es SEAlienigenaVolador pero su tipo estática es SEVolador

2. Indica en qué líneas no hay error, hay error de compilación o hay error de ejecución.

```
SuperHeroe sh2= new SEAlienigenaVolador("capitanZ", "Smirk", 3.8);
```

No hay error de ningún tipo

```
SEVolador sh3= new SEVolador("Anaceto", 1);
```

Error de Compilador. No se puede instanciar una clase abstracta.

```
ControlDePoder sh4= new Alienigena("ET", "Micasa");
```

No hay error de ningún tipo, se pueden crear objetos a partir de una interfaz mientras se inicialicen a otro tipo que la implemente.

```
SEVolador sh5= new SETerrestreVolador("JunLee", 2);
```

No hay error de ningún tipo.

```
sh5= SEAlienigenaVolador("Crushi", "Crush", 200);
```

Error de compilación, falta el new

```
ArrayList<SEVolador> coleccion= new ArrayList<>();
```

No hay problema, pero dentro del array solo pueden introducirse objetos de subclases de SEVolador

```
coleccion.add(sh5);
```

Se puede hacer sin problema.

```
coleccion.add(sh2);
```

Error de compilación, ya que el tipo estático de sh2 es SuperHeroe, y no se puede meter en el array un objeto de una superclase.

```
coleccion.get(0).alunizar();
```

Error de compilación, el tipo estática de sh5 es SEVolador.

3. Implementar en Ruby el método rescatar de SEVolador para que llame a volar y si la altura del vuelo es mayor de 50 llame luego al método rescatar de la superclase.

```
def rescatar
    volar()
    if @alturaVuelo > 50
        super
    end
end
```

4.Implementa en Java las siguientes clases e interfaces completas, incluyendo la implementación interna de los constructores que aparezcan en el diagrama:

- SuperHeroe
- ControlDePoder (donde el método informar devuelve el string "tengo el poder")

```
Public class SuperHeroe implements ControlDePoder {
    Public Static String Sindicato = "SSE-SA";
    private String Nombre;
    protected boolean estadoPoderes;
    private ArrayList<Humano> rescatados = new ArrayList<Humano>();
    private ArrayList<Poder> poderes = new ArrayList<Poder>();
    public static SuperHeroe (String nombre) {
        this.nombre = nombre;
    }
    private string getNombre () { return this.nombre; }
    public void addPoder (Poder poder) { poderes.add (Poder); }
    public boolean getEstadoPoderes () { return this.estadoPoderes; }
    public String rescatar (Humano humano) {...}
    @Override
    public void PoderesON () {...}
    @Override
    public void poderesOFF () {...}
}

public interface ControlePoder {
    public abstract void poderesON () {};
    Public abstract void poderesOFF () {};
    pubic String informar () {};
}
```

5. Crea una clase parametrizada llamada Librería que permita gestionar libros de diferentes tipos. Como atributos tiene el nombre de tipo String y una lista de libros del tipo del parámetro. Crea un método para añadir un libro que recibe como parámetro en la primera posición de la lista libros y otro para consultar el último libro de dicha lista. La cabecera de la clase sería:

```
public class Libreria<T extends Libro> {

    private String tipo ;

    private ArrayList <String> libros = new ArrayList <String> () ;

    Public Libreria ( String tipo ) {

        this.tipo = tipo ;


    }

    public void addLibro ( String libro) {

        Libros . add ( libro) ;
    }

    private String getUltimoLibro () { return libros.get(0) ; }
}
```

6. Dada la siguiente declaración en una clase del mismo paquete que las del diagrama:

```
SuperHeroe sh3= new SEAlienigenaVolador("R74", "Marte", 1.5);
```

Indica en qué líneas hay error de compilación o de ejecución y cómo se corregirán:

```
sh3.getNombre();
```

Error en tiempo de ejecución, sh3 al ser un objeto SEAlienigenaVolador no puede acceder al consultor de nombre de SuperHeroe ya que es privado, y tendría que hacerse público.

```
sh3.getEstadoPoderes();
```

Se ejecuta sin problema, no hay ningún error.

```
sh3.alunizar();
```

Error de compilación. Se corregiría de la siguiente forma : ((SEAlienigenaVolador) sh3).alunizar();

```
((SETerrestreVolador)sh3).golpear();
```

Error de compilación. No se puede hacer un casting entre hermanos.

En todo caso sería : ((SEForzudo) sh3).golpear();

7. Indica cómo harías para hacer copia profunda de los objetos de la clase SuperHeroe.

```
public class SuperHeroe implements ControlDePoder, Cloneable {

    private ArrayList<Humano> rescatados = new ArrayList<Humano>();
    private ArrayList<Poder> poderes = new ArrayList<Poder>();

    public void ArrayHumanos (ArrayList<Humano> h) {
        rescatados = h;
    }
    public void ArrayPoderes (ArrayList<Poder> p) {
        poderes = p;
    }

    @Override
    public ArrayHumanos clone() throws CloneNotSupportedException {
        return (ArrayHumanos) super.clone();
    }
    @Override
    public ArrayPoderes clone() throws CloneNotSupportedException {
        return (ArrayPoderes) super.clone();
    }
}
```