



1. Indica verdadero o falso:

- La relación entre ControlDePoder y SuperHeroe es de dependencia

Falso, ControlDePoder es una interfaz y SuperHeroe realiza esa interfaz.

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

Falso, en UML, la cursiva representa que dicha clase es una clase abstracta.

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

Falso, el diagrama representa que "cada Superhéroe tiene su propio poder"

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

Podemos declarar la constante pero no podrá ser protegida, pues las constantes son, por defecto, public static final en interfaces

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

Verdadero, de hecho si no lo estuviera, las clases hijas deberían ser abstractas.

- El método informar está sobrecargado en SEForzudo

Falso, está redefinido al heredarlo de SuperHeroe y questa clase realiza ControlDePoder

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

Verdadero, pues para ello debe ser de tipo SuperHeroe

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

Si no fuera abstracta esto sería cierto pero como no se puede instanciar su clase

abstracto es falso

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

Falso, hereda todos pero sólo tiene acceso a los declarados en su clase.

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

Falso, podemos declarar una clase como abstracta sin tener métodos abstractos, la única funcionalidad es evitar que se instancien objetos de este tipo.

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

Desconozco la respuesta

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

Verdadero, pues no sabíamos cuál se hereda, esto haría obligatoria redefinir si se hace o no el diseño

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

Cierto, heredado desuperheroe

- En el siguiente código:

```
SuperHeroe batman= new SETerrestreVolador("Batman", [REDACTED] 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

Cierto, en este caso, se ejecutará el código asociado a la clase SEVolador

- 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, según el diagrama de clases, el consultor de nombre es privado.

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

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

Falso, se ejecuta perfectamente por tener acceso 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, el tipo estático es SEVolador

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

SuperHeroe sh2 = new SEAlienigenaVolador("capitan","Smirk",3.8); *No hay error*
SEVolador sh3 = new SEVolador("Anacleto",1); *No hay error*
ControlDePoder sh4 = new Alienigena("ET","micasa"); *Error de compilación, sh4 no puede referenciar a algejobjeto que no realiza la interfaz*
SEVolador sh5 = new SETerrestreVolador("Junlee",2); *No hay error*
sh5 = SEAlienigenaVolador("Crushi","Crush",200); *Error de compilación, falta la palabra reservada new*
ArrayList<SEVolador> colección = new ArrayList<>(); *No hay error*
colección.add(sh5); *No hay error*
colección.add(sh2); *Error de compilación pues un SuperHeroe no es un SEVolador*
colección.get(0).alunizar; *Suspenso que linea 5 se ejecuta no hay fallo, en otro caso, bug error de compilación pues SEVolador no dispone del método alunizar.*

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 (humano)
    volar
    if @alturaVuelo > 50 then
        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") → trivial

```
public class SuperHeroe implements ControlDePoder {
    public static String Slogan = "SSE-SA"; private ArrayList<Humanos> rescatados;
    private String nombre;
    protected boolean estadoPoderes;
    private ArrayList<Poderes> poderes;
    public SuperHeroe (String nombre) {
        nombre = nombre;
        estadoPoderes = true;
        poderes = new ArrayList<>(); rescatados = new ArrayList<>();
    }
    private String getNombre () {
        return nombre;
```

```
public void actoPoder (Poder poder) {
    poderes.add (poder);
}

public boolean getEstadoPoderes () {
    return estadoPoderes;
}

public String rescatar (Huwano huwano) {
    rescatados.add (huwano);
    return "El huwano " + huwano.nombre + " ha sido rescatado por " + nombre;
}
```

@Override
public void poderesON ()

estadopoderes = true;

}

@Override

```
public void poderesOFF () {
    estadopoderes = false;
```

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

```
public class SuperHeroe implements ControlDePoder, Clonable {
    private nombre,
    protected estadoPoderes;
    private ArrayList<Huwano> rescatados;
    private ArrayList<Poderes> powerups;
```

```
public ArrayList<Huwano> getrescued () {
    ArrayList<Huwano> copia = new ArrayList<>();
    Huwano h=null;
    for (Huwano h:rescued) {
        try {
```

h = h.clone(); //supongamos que Huwano realiza Clonable

catch (CloneNotSupportedException) System.out.println ("Error al clonar");

```
        copia.actel(h);
    }

    return copia;
}

public ArrayList<Poder> getPowers() {
    ArrayList<Poder> copia = new ArrayList<>();
    Poder h=null;
    for(Poder haw: powerups) {
        try {
            h = haw.clone(); //suponemos Poder realiza Clonable
        catch(CloneNotSupportedException) System.out.println("Error al clonar");
    }
}
```

```
        copia.actel(h);
    }

    return copia;
}
```

@Override

```
public SuperHeroe clone() throws CloneNotSupportedException {
    SuperHeroe copia = (SuperHeroe) super.clone();
    copia.rescued = this.getRescued();
    copia.powerups = this.getPowers();
}
```