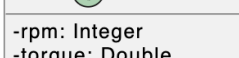


[illegible]

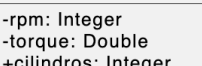
1) Para la clase descrita en la Figura 1, determine la cantidad de métodos setters y getters que se necesitan. **(1pts)**

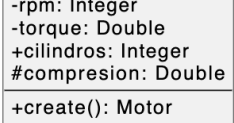


```

classDiagram
    class Motor {
        -rpm: Integer
        -torque: Double
        +cilindros: Integer
        #compresion: Double
        +create(): Motor
        ..otros métodos
    }
        
```

Figura 1

<p>2 Para la clase descrita en la Figura 2, de un ejemplo de <i>declaración</i> de setter. (1pts)</p>	<div data-bbox="1018 892 1218 957">  <pre> classDiagram class Motor { -rpm: Integer -torque: Double +cilindros: Integer #compresion: Double +create(): Motor ..otros métodos } </pre> <p>Figura 2</p> </div>
<p>+ setRPM(r: integer): void</p> <p>Control de acceso: 0.3pts (sólo con definición correcta)</p> <p>Definición correcto: 0.7pts</p>	

<p>3) Para la clase descrita en la Figura 3, de un ejemplo de <i>declaración</i> de getter. (1pts)</p>	<div data-bbox="1003 1396 1235 1518">  <pre> classDiagram class Motor { -rpm: Integer -torque: Double +cilindros: Integer #compresion: Double +create(): Motor ..otros métodos } </pre> </div> <p>Figura 3</p>
<p>+ getRPM(): integer</p> <p>Control de acceso: 0.3pts (sólo con definición correcta)</p> <p>Definición correcto: 0.7pts</p>	

4) Considere el diagrama de la Figura 4. Determine si la siguiente instrucción es válida o no. (2pts)

```
Persona p0 = CREATE_OBJECT Persona()
```

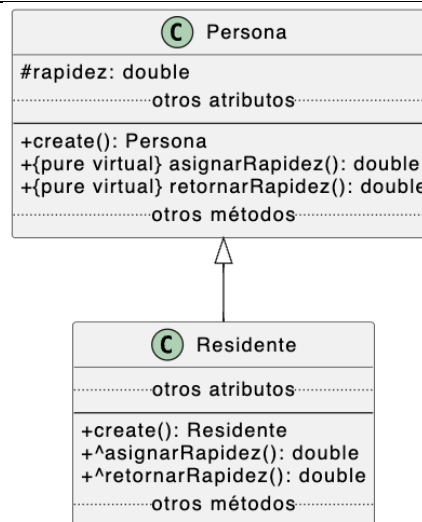


Figura 4

Un método virtual puro es un método que está declarado, pero que no admite implementación en la clase base. Debido a esto, una clase que tiene por lo menos un método virtual puro, no puede instanciar directamente un objeto.

Por lo tanto, **la instrucción no es válida**.

5) Considere el diagrama de la Figura 5. Determine si la siguiente instrucción es válida o no. (2pts)

```
Persona a0 = CREATE_OBJECT Residente()
```

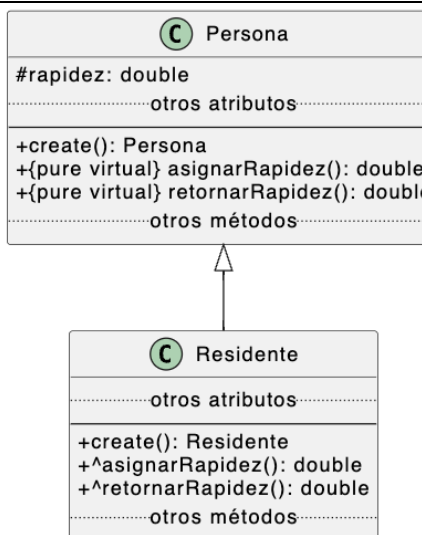


Figura 5

En este caso, se está creando un objeto Residente. La estructura de herencia indica un Residente ES una persona. Debido a esto, la instrucción es válida.

6) Considere el diagrama de la Figura 6. Determine si la siguiente instrucción es válida o no. (2pts)

Residente a0 = CREATE_OBJECT Residente()

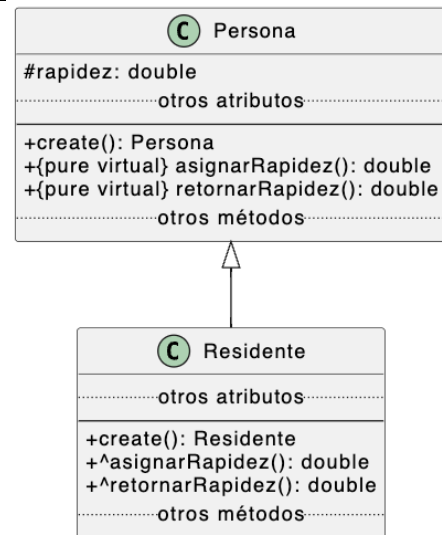


Figura 6

En este caso, se está creando un objeto Residente y la variable es del mismo tipo. Por lo tanto, la instrucción es válida.

8) Considere el diagrama de la Figura 7. Implemente en pseudo-código la clase **Residente**. (3pts)

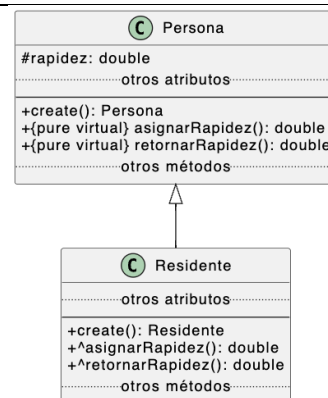


Figura 7

```

class Residente:

    public:
        create() {
            //Código constructor
        }

        OVERRIDE void asignarRapidez(r: double) {
            rapidez = r
        }

        OVERRIDE double retornarRapidez() {
            return rapidez
        }
    
```

No es necesario que el pseudo código sea idéntico. Sólo debe expresar las mismas ideas.

9) Considere el diagrama de la Figura 8. Además, se implementa la siguiente función:

```
FUNCTION mostrarPeso(Animal a): Void  
    print(a.retornarPeso())
```

y se crea el siguiente objeto:

```
Animal a0 = CREATE_OBJECT Perro(17.5)
```

Determine si la siguiente instrucción es válida o no: `mostrarPeso(a0)`. (3pts)

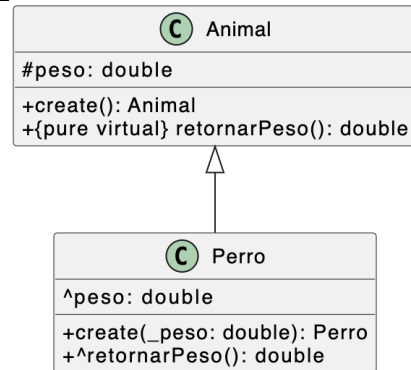


Figura 8

Desde el punto de vista de herencia, la clase Perro es derivada de la clase Animal. Por lo tanto, desde el punto de vista de POO, un objeto de la clase Perro es también un objeto de la clase Animal. **Luego, la llamada a la función es válida.**

10) Considere el diagrama de la Figura 9. Suponga que las clases **Animal** y **Perro** están correctamente implementadas. Se crean una instancia de cada clase:

```
Animal a0 = CREATE_OBJECT Perro(3.5)  
Perro p0 = CREATE_OBJECT Perro(4.5)
```

Determine si las siguientes asignaciones son válidas o no: (3pts)

```
a0.peso = 3.7  
p0.peso = 4.8
```

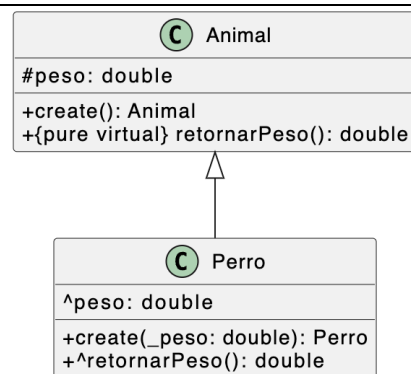


Figura 9

Ambas no son válidas, ya que el atributo peso es protected y no puede ser accedido desde fuera de la jerarquía de la clase donde fue declarado.