



1. Supón que todas las clases del diagrama tienen un constructor que inicializa todos sus atributos utilizando valores que le llegan como argumento. Implementa en Ruby la cabecera de la clase Pierrot y su constructor.

Jara

Ruby:

module Circo

```

class Pierrot < Payaso
    def initialize (uArtístico, uPila, llevoS, duracionJ)
        super (uArtístico, uPila, duracionJ)
        @llevaSoub = llevoS
    end

```

end

end

Java:

```

package Circo;
public class Pierrot extends Payaso {
    private boolean llevaSoub;
}

```

public Pierrot (String pila, String artístico, boolean llevoS, int duracionGira){
super (pila, artístico, duracionGira);
this. llevaS = llevoS;

{

{

TRES. LLEVASUNDERRU = llevoS;

2. Implementa en Ruby los métodos salirAEscena de Payaso y de Augusto de modo que:

- en Payaso imprima por pantalla "¿Cómo están ustedes?"
- en Augusto haga lo que hace un Payaso y a continuación haga una travesura

module Circo

class Payaso < Artista

@@ Salida = "¡Qué estás ustedes?" ← Se sobrescribe si lo cambian alguno

def salirAEscena

puts @@ Salida

end

end

class Augusto < Payaso

def salirAEscena

super

hacerTravesura

end

(private.)

def hacerTravesura

puts "Hago travesuras"

end

end

end

Se puede poner privado pero puede hacer travesuras después de salirAEscena.

3. Supón que tanto en Artista como en la subclase Trapecista tenemos los siguientes métodos de clase:

```
public static void setDuracionGira(int d){ duracionGira=d; }  
public static int getDuracionGira(){ return duracionGira; }
```

¿Qué se imprimiría con las siguientes líneas de código?

¿Qué se imprimiría con las siguientes líneas de código?

```
Asumiendo que Artista = new Trapecista(...);  
Artista.setDuracionGira(2);  
System.out.println(Artista.getDuracionGira());
```

Claramente devolverá 2, pues un Trapecista es un artista y como no tiene la implementación para Trapecistas busca en la clase padre hasta encontrar una implementación.

```
System.out.println(Trapecista.getDuracionGira());
```

De nuevo, como Trapecista no tiene implementación pues la busca en el método y de la misma manera, la busca en artista dando 2.

```
Trapecista.setDuracionGira(4);  
System.out.println(Artista.getDuracionGira());
```

De ejercicios anteriores `Artista.getDuracionGira()=2` y como Trapecista no es Artista como instancia, obtendremos un 2.

```
System.out.println(Trapecista.getDuracionGira());
```

Por la justificaciónanáloga, obtendremos 4.

¿Y si la subclase tuviese su propio atributo de la clase duracionGira inicializado a 4?

```
Artista.setDuracionGira(2);  
System.out.println(Artista.getDuracionGira());
```

Como Artista no es un Trapecista, obtendremos un 2.

```
System.out.println(Trapecista.getDuracionGira());
```

Si dispone de implementación dentro de la clase, se obtendrá un 4; en otro caso, asumiendo que acróbata no tiene se ejecutará la de artista obteniendo un 2.

```
Trapecista.setDuracionGira(4); ← Búlgio redondeando  
System.out.println(Artista.getDuracionGira());
```

Como llamamos a artista obtendremos un 2.

```
System.out.println(Trapecista.getDuracionGira());
```

La respuesta dependerá de si hay implementación o no al igual que los ejemplos anteriores.

4. Supón que en la clase Pierrot existen los siguientes métodos adicionales donde compi es otro objeto Pierrot:

Ruby:

```
def presentacionCompi(compi)
    puts "Hoy está conmigo #{compi.nombreArtistico} vestido de
        #{compi.colorAtuendo}"
end
```

Java:

```
void autoPresentación(Pierrot compi){
    System.out.println("Me llamo " + nombreArtistico + " y voy vestido
de " + colorAtuendo);
}
```

Corrige los fallos en los métodos (si los hay). Si es necesario hacer algún cambio en alguna clase indica.

Suponemos que los getters nombreArtistico y colorAtuendo están implementados pues en caso contrario, no podríamos acceder en Ruby a los atributos privados de compi. Esto es así por la visibilidad privada de Ruby.

En Java sí podemos acceder, no obstante, se accede a los del objeto implícito, debería poner compi.nombreArtistico y compi.colorAtuendo.

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

```
Artista x1 = new Acrobata();
```

Un Acrobata es un artista luego no hay fallo, habrá limitaciones en las cuales usar por el tipo estático.

```
Gimnasta x2 = new Funambulista();
```

Un Funambulista es un gimnasta luego no hay fallo; sin embargo, volverá a tener restricciones por el tipo estático.

```
Payaso x3 = new Payaso();
String s = x3.getNombreArtistico();
```

No hay fallo pues se ejecutará el método de artista.

```
Acrobata x4 = new Trapecista();
x4.agarrar();
```

Trapecista no dispone (o no sabe) del método agarrar, nos dará un fallo de compilación pues para el compilador, x4 es un Acrobata.

```
Payaso pa2 = new Pierrot();  
((Augusto) pa2).salirAEscena();
```

Da fallo de ejecución pues, para debe ser un Augusto; sin embargo, para es un Pierrot por el tipo dinámico.

```
Artista pa3 = new Augusto();  
((Payaso) pa3).salirAEscena();
```

No hay ningún fallo pues estamos realizando un upcast.

6. Supón que en nuestro sistema hay algunos magos mentalistas que no hacen trucos como cualquier otro mago, sino que leen la mente. Además, son capaces de hipnotizar, cosa que no cualquier mago puede hacer.

¿Cómo incluirías estas novedades en el sistema?

Ruby

module Magia

```
class Mentalista < Mago
```

```
def initialize(atributo_magia)
```

```
super
```

```
end
```

```
def hacerTruco(persona) #supongo que lo hace Mago yendo — @Override
```

```
puts "Leer Mente"(persona)
```

```
end
```

```
def hipnotizar(persona)
```

```
puts "Te estoy hipnotizando persona vacubre"
```

```
end
```

```
end
```

```
end
```

Java

package Magia;

```
public class Mentalista extends Mago {
```

```
public Mentalista(atributo_magia){
```

```
super(atributo_magia);
```

```
{
```

```
def hacerTruco(persona) #supongo que lo hace Mago yendo — @Override
```

```
public void hacerTruco(Persona p){
```

```
System.out.println("Leer Mente");
```

```
}
```

```
public void hipnotizar(Persona p){
```

```
System.out.println("Te estoy hipnotizando persona vacubre");
```

```
{
```

```
}
```