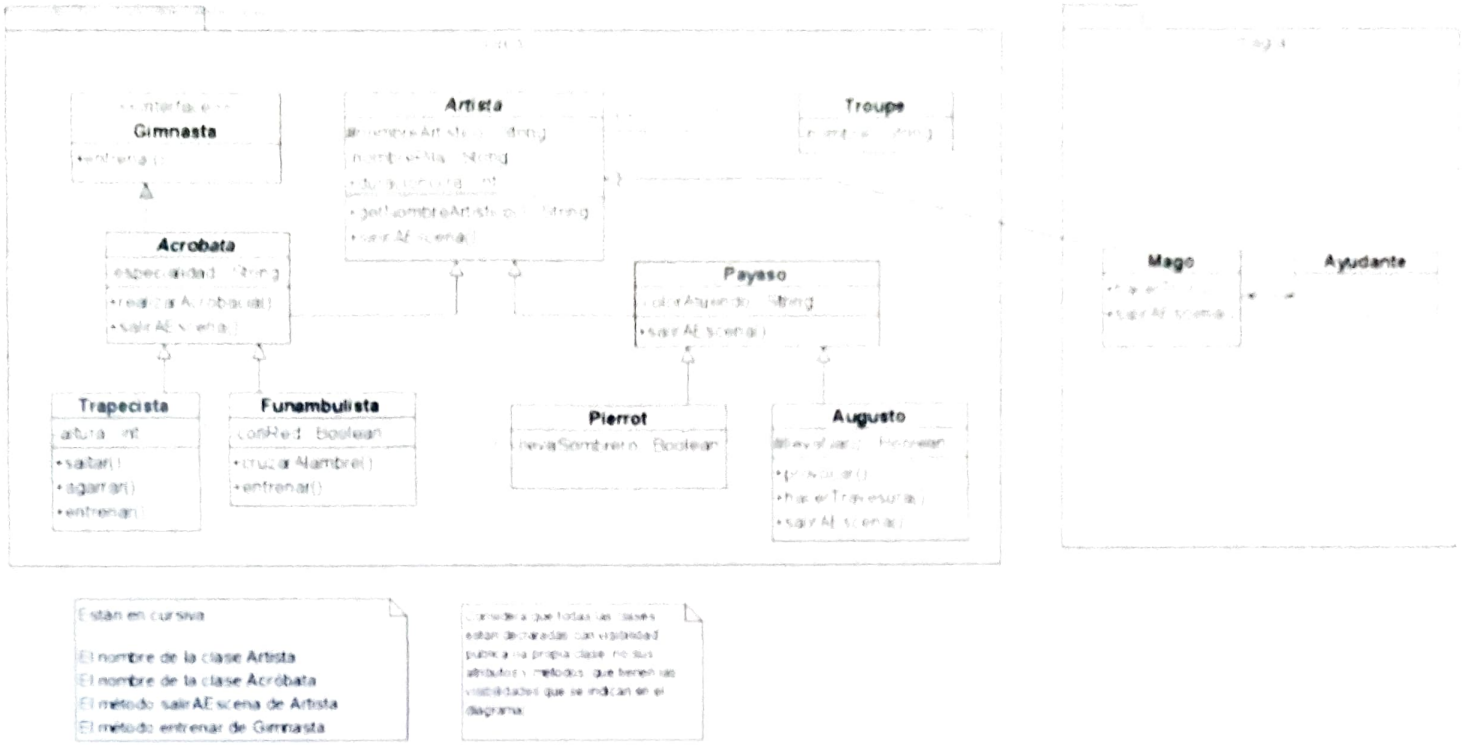

Están en cursiva

El nombre de la clase Artista
El nombre de la clase Acróbata
El método salirAEscena de Artista
El método entrenar de Gimnasta

Considera que todas las clases están declaradas con visibilidad pública y la propia clase no sus atributos y métodos que tienen las visibilidades que se indican en el diagrama.

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~~ Java la cabecera de la clase Pierrot y su constructor.**

```
package circo
public class Pierrot implements Payaso {
    private boolean llevaSombrero
    public Pierrot ( boolean llevaS , String colorAtuendo, String nArtistico , String nPila ,
                    int duracionGira ) {
            super( colorAtuendo , nArtistico , nPila , duracionGira ) ;
            this.llevaSombrero = llevaS ;
    } }
```

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**

```
class Payaso < Artista
  ...
  def salirAEscena
      tmp = super
      tmp += " ¿Cómo están ustedes ? "
      puts tmp
  end
  ...
end
class Augusto < Payaso
  ...
  def salirAEscena
        tmp = super
        tmp += hacerTravesura ()
        puts tmp
  end
end
```

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?

Asumiendo que Artista = new Trapecista(...);

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

2

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

2

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

2

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

4

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

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

2

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

4

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

2

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

4

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 indícalo.

Ruby: - Hay que crear consultores para poder acceder a las variables nombreArtistico y color atuendo ⟶ attr_reader : nombreArtistico , colorAtuendo

Java: - Habría que crear consultores para poder acceder a las variables usados :

nombreArtistico tiene visibilidad de paquete

public String getColorAtuendo () { return colorAtuendo;}

- No hay que pasar ningún argumento a la función

Ejercicio 1 : Ruby

```
module Circo
class Pierrot < Payaso
    @llevaSombrero
    def initialize ( llevaSombrero, colorAtuendo , nombreArtistico , nombrePila )
          super( nombreArtistico, nombrePila , colorAtuendo)
          @llevaSombrero = llevaSombrero
    end
...
end
```

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

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

Error de compilación ⟶ Acróbata también es abstracta

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

Sin errores

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

Sin errores

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

Error de compilación ⟶ se puede resolver mediante casting : ((Trapecista) x4).agarrar() ;

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

Error de compilación ⟶ no se puede hacer casting entre hermanos (Pierrot y Augusto)

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

Sin errores

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?

Java :

```
package magia
import java.circo.Artista
class Mentalista extends Mago {

    Mentalista (String nombreArtístico,
                String nombrePila) {
        super(nombreArtístico, nombrePila);
    }
    @Override
    public void hacerTruco() {
        tmp = super()
        + System.out.println("El mago te está
            leyendo la mente");
    }
    @Override
    public void hipnotizar() {
        System.out.println("One, two, Django!");
    }
}
```

Ruby :

```
module magia
require_relative /circo/Artista.rb

class Mentalista < Mago

    def initialize(nombreArtistico, nombrePila)
        super(nombreArtistico, nombrePila)
    end

    def hacerTruco
        tmp = super
        tmp += "El mago te está leyendo la mente"
        puts tmp
    end
    def hipnotizar
        tmp = "One, two, Django!"
        puts tmp
    end
```