Ejercicio de análisis Taller 7 POO

1

a) Debe definirse como clase abstracta ya que tiene definidos métodos abstractos. A diferencia de una clase normal, el constructor debería ser protegido ya que no es posible instanciar un objeto que sea únicamente de tipo **Instrumento**

b) public class Piano extends Instrumento {

```
public Piano(String tipo) { super(tipo); }
public void tocar () {System.out.println(“Tocando piano”);}
public void afinar() {System.out.println(“Afinando piano);}

}
```

c) Todas las líneas se pueden ejecutar. Si bien es cierto que no se pueden instanciar objetos de una clase abstracta, sí que se pueden definir apuntadores del tipo de la clase (x en este caso), ya que ese podrá apuntar a objetos de las subclases de esta, como lo hace cuando se instancia un objeto de tipo **Saxofon**, para luego perder este apuntador, ya que ahora se encontraría apuntando a un objeto de tipo **Guitarra**

d) “Tocando saxofon”
“Tocando guitarra

2

a) No es posible hacer esto, ya que el método **explotar()** es abstracto; no es posible definir el cuerpo del método

b) No es un error, esto simplemente significa que estos métodos serán heredados por las subclases, pero, a diferencia de lo que sucede con los métodos abstractos, estos métodos no será necesario definirlos porque ya están definidos desde la clase padre.

c) No es un error, el propósito de una clase abstracta es contener características comunes entre las clases que van a heredar de esta. Si no se definen métodos abstractos, las clases hijas heredarán los métodos no abstractos previamente definidos, pero la funcionalidad de la clase abstracta sigue intacta

d) Esto es con el objetivo de poder almacenar varios objetos derivados de esta clase en este arreglo, ya que todos son de tipo **ObjetoAstronomicoExtrasolar**. En la línea 25 se llama al método específico para cada elemento debido a la ligadura dinámica por la que se resuelven los métodos; se ejecuta el método de la clase más específica

e) Porque en la línea 28, el objeto inicializado como galaxia pierde su apuntador, ya que **oa[0]**, el cual apuntaba a este objeto, ahora pasa a

apuntar al mismo objeto que se encuentra en la posición **oa[2]**; una supernova, o sea que llamar a **oa[0].descripcion()** es lo mismo que llamar a **oa[2].descripcion()**

f) Esto se debe a que la clase **Estrella** también es abstracta, por lo que no necesita definir todos los métodos abstractos de su clase padre, cosa que sí tendrán que hacer las subclases de **Estrella** (definir los métodos abstractos de **ObjetoAstronomicoExtrasolar** y los de **Estrella**)

g) Porque **tipoCuerpo1()** es heredado de **ObjectoAstronomicoExtrasolar**, y esta clase tiene declarado este método con visibilidad por defecto. Genera error porque no se puede cambiar la visibilidad de un método heredado a una más restrictiva, cosa que sucedería si **Galaxia** definiera este método como privado. En términos de visibilidad, package>private

h) No es necesario definirlo ya que su clase padre (**Estrella**) ya lo definió. Si se volviese a definir este método, se estaría sobre escribiendo la definición de la clase padre, la cual ahora sería accesible mediante el modificador **super**

i) Imprime lo mismo que si se llamara a System.***out***.println(g), ya que no se ha redefinido este método. Se puede llamar a este método ya que es heredado de la clase **Object**

j) Las clases abstractas impiden la creación de objetos que sean únicamente de esta clase, pero no hay restricción a la hora de crear punteros, ya que se puede tratar a un objeto de una subclase de una clase abstracta como si fuera del tipo de la clase abstracta gracias al polimorfismo

k) B no es válida, ya que no es posible instanciar un objeto de una clase abstracta que no pertenezca a una de sus subclases. C sí es válida, ya que se crea un puntero de tipo **ObjetoAstronomicoExtrasolar** que apunte a un objeto de tipo **Nova**, una subclase de **ObjetoAstronómicoExtrasolar**, por lo que se puede tratar a nova como si fuera de este tipo sin problemas

l) La instrucción en la línea B es correcta porque es la misma línea C del punto anterior. La línea C es incorrecta ya que, si bien los métodos se resuelven por ligadura dinámica, yendo a la definición más específica, para que esto sea posible, primero se debe tener una definición de este método para la clase del tipo del apuntador, cosa que no sucede en este caso; **ObjetoAstronomicoExtrasolar** no tiene definido el método **explotar().** Ignorando la instrucción de la línea C, en este caso se imprime: “Boom!” por pantalla, ya que este es un caso correcto de especificación o downcasting, donde se quiere tratar a un objeto con un apuntador definido de una clase superior, como si fuese de un tipo inferior (subclase) a esta clase. Esto es posible ya que, en efecto, el objeto al que se le quiere realizar el downcasting de **ObjectoAstronomicoExtrasolar** a **Nova** sí que es un objeto de tipo nova

m) Porque todos los objetos, sin importar su tipo de apuntador o clase a la que pertenecen, heredan y son instancias de **Object**, porque todas las clases

heredan de esta. Por esta razón, para cualquier objeto, esta línea debería imprimir true.

La primera línea imprime false, ya que **null** quiere decir que ese apuntador no apunta a nada, o sea que no apunta a ningún objeto, pero en la segunda línea imprime true, ya que se está convirtiendo el valor de la referencia de **obN** en un String, y String sí hereda de Object

n) No se genera error pero este no puede ser usado directamente; tendrá que ser usado desde el constructor de alguna de las subclases. Este es el propósito de definir un constructor para una clase abstracta: que lo usen las clases hijas. Por esta razón es preferible que sea definido como **protected**

o) Se generan errores porque **EnanaBlanca** hereda de **Estrella**, la cual es una clase abstracta que a su vez hereda de **ObjetoAstronomicoExtrasolar**, por lo que **EnanaBlanca** debe definir los métodos abstractos presentes tanto en **Estrella** como en **ObjetoAstronomicoExtrasolar**