Ejercicio de análisis

a) Explique por qué la clase Instrumento debe definirse como abstracta y qué la diferencia de una clase normal, en el ejemplo siguiente: La clase Instrumento sirve como una especie de plantilla para las clases que heredan de ella, como lo son Saxofon y Guitarra, es por esto que debe ser abstracta, ya que no va a ser invocada directamente, sino que será uasada a través de sus subclases. Instrumento genera unas especificaciones que deben tener todas las clases que hereden de ella, como el hecho de que se deba añadir un atributo tipo y que se pueda acceder con un método get usando `super`; además, obliga a que se definan los métodos Tocar y Afinar.

La diferencia con una clase normal, es que esta última puede ser invocada directamente, y todos sus métodos deben estar completamente definidos.

b) Elabore una clase denominada Piano heredada de Instrumento, añada un constructor y redefina para ella los métodos Tocar y Afinar.

```
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) ¿Señale cuál de las siguientes líneas no se pueden ejecutar y por qué? Todo el código se ejecuta sin problemas, primero se crea un apuntador de la clase Instrumento, luego se crean dos objetos Saxofon y Guitarra (subclases de Instrumento), primero se le asigna al apuntador un objeto y después el otro.

d) ¿Qué imprime el siguiente programa? Imprimirá dos líneas: la primera, cuando la referencia X apunte a un objeto de la clase Saxofon, donde se resuelve la ejecución del método Tocar por ligadura dinámica, y se imprime "Tocando Saxofon"; la segunda, ocurre cuando la misma referencia X pasa a puntar un objeto de la clase Guitarra, y se ejecuta el método Tocar de forma similar a la clase Saxofon, resolviendo por ligadura dinámica y devolviendo "Tocando Guitarra".

Ejercicio de código en el repositorio

a) ¿Qué sucede si se define el método explotar() de la clase Estrella como se indica a continuación? Explique su respuesta. Existirá un error en compilación, pues no se puede definir un método abstracto, pero en el método abstracto explotar se está intentando imprimir un mensaje en pantalla.

b) ¿Qué significa que los métodos tipoCuerpo2() y getID() de la clase ObjetoAstronomicoExtraSolar, no se definan como abstract? ¿Podría considerarse esta situación un error? Explique. Definir métodos en una clase abstracta no incurre en erorres de compilación o ejecución, por lo contrario, pueden resultar útiles para crear métodos generales que puedan usarse entre las clases que hereden de ella. Lo que no es posible es definir un método abstracto, pero definir un método en una clase abstracta es otro asunto.

c) Si se define como abstracta la clase ObjetoAstronomicoExtraSolar, como se indica a continuación, ¿puede considerarse un error definir una clase abstracta sin métodos abstractos? Explique. Si bien no es un error en términos de ejecución y compilación, hacer una clase abstracta donde tanto todos los métodos como atributos están definidos no resulta a priori un diseño deseable. Esto es así porque si ninguna parte del código de la clase es abstracto, ésta podría ser definida directamente como una clase normal y facilitar su implementación a la hora de crear instancias sin necesidad de subclases. #### d) Explique por qué el arreglo oa (línea 19) hace referencia a una clase abstracta y sin embargo, en la línea 25 se invoca el método correspondiente a cada clase derivada.

Las referencias a clases abstractas permiten hacer llamados a métodos comunes de las subclases. En este caso, descripcion existe en ObjetoAstronomicoExtraSolar como un método abstracto, que debe ser definido en todas sus clases hijas, ya cada una tiene la libertad de devolver la cadena que precise.

e) ¿Por qué la línea 29 imprime "Soy una Super Nova" sabiendo que el arreglo oa en esa posición fue inicializado con un objeto de tipo Galaxia? Imprime ¨Soy una Super Nova" porque antes de ejecutar la instrucción de impresión, se produjo una reasignación en la lista de la siguiente manera:

```
oa[0] = oa[2];
//a la primera posición de la lista oa se le asigna el valor guardado en su segunda posició
```

Volviendo atrás en el código, se puede evidenciar que la segunda posición almacenaba un objeto de tipo SuperNova, por tanto, el método descripción aplicado a la primera posición, también es el de la clase SuperNova.

```
oa[2] = new SuperNova();
```

f) ¿Por qué en la clase Estrella no se define el método descripcion() si la superclase lo está solicitando, ya que en este método descripción en abstracto? Estrella no incurre en problemas al no agregar el método descripción, ya que es también una clase abstracta, lo que significa que si bien sigue heredando la obligación de que sus subclases no abstractas deban definir descripción, no es necesario reescribirlo explícitamente.

g) ¿Qué sucede si el método tipoCuerpo1() de la clase Galaxia se define como privado? ¿Por qué se genera error? Genera error debido a que la línea 8 llama explícitamente a la función fuera del paquete de su clase, y como ahora pasó de ser pública a privada, ya no resulta accesible.

```
g.tipoCuerpo1(); //linea que genera el error
```

```
The method tipoCuerpo1() from the type Galaxia is not visible
```

h) ¿Por qué la clase Nova no define el método tipoCuerpo1()? ¿Se podría definir? Si lo define, ¿qué interpreta de esta situación? Nova no necesita definir el método tipoCuerpo1 porque dentro de su clase padre (Estrella) ya está completamente definido:

```
public void tipoCuerpo1() {

System.out.println("Simple " + a);

}
```

Sería un caso distinto si el método fuese abstracto; en este caso simplemente se hereda. De todas formas, se podría cambiar el retorno de la función usando @Override en la clase Nova, de la siguiente manera:

```
@Override
public void tipoCuerpo1() {

System.out.println("Nova Simple " + a);

}
```

i) ¿Qué imprime la línea 9? ¿Por qué se puede llamar al método toString() si la clase Galaxia no lo define y su papá ObjetoAstronomicoExtraSolar tampoco? La línea 9:

```
System.out.println(g.toString());
```

Se puede ejecutar porque toString es una función que viene integrada en la clase Object y toda su descendencia, es decir, en orden jerárquico se hereda de esta manera:

Object -> ObjetoAstronomicoExtraSolar -> Galaxia

j) . ¿Por qué en la línea 11 se puede crear un puntero obN de tipo ObjetoAstronomicoExtraSolar si esta es una clase abstracta? No existe ningún problema con crear punteros de tipo de clases abstractas, de hecho, puede ser útil para generar listas de subclases de una clase abstracta dada, lo que no es posible es crear un objeto de la clase ObjetoAstronomicoExtraSolar, pero lo

que se está haciendo en el código es hacer un cast explícito de un objeto de clase Nova a su clase abstracta más general.

```
ObjetoAstronomicoExtraSolar obN = (ObjetoAstronomicoExtraSolar) nova;
```

k) ¿Las siguientes instrucciones (instrucciones en las líneas B y C) son válidas? Explique La instrucción A es legítima, simplemente crea un objeto de la clase Nova y lo referencia con un puntero de la misma clase (no abstracta).

La instrucción B no es válida, porque intenta crear un objeto de una clase abstracta, por lo que ocurrirá un error de compilación.

La instrucción C también es posible, siempre y cuando oa no sea una variable ya existente, ya que se puede hacer una referencia de tipo clase abstracta que apunte a una subclase definida. Si se intenta ejecutar con `ObjetoAstronomicoExtraSolar[] oa` ya definida, se incurrirá en error de duplicación.

l. Explique por qué (ver código a continuación) la siguiente instrucción en la línea B es correcta y la instrucción en la línea C es incorrecta. Omitiendo la instrucción en la línea C, ¿qué se imprime por pantalla? Explique su respuesta. La instrucción B es correcta porque Nova es de tipo ObjetoAstronomicoExtraSolar, y porque es posible hacer una referencia de tipo abstracto mientras sea la clase padre.

La instrucción C es incorrecta porque el método explotar no fue un método (abstracto o definido) que haya sido enunciado dentro de la clase ObjetoAstronomicoExtraSolar, por lo que el compilador de Java no encuentra un método para ejecutar, ya que se guía solamente de que la referencia es de tipo ObjetoAstronomicoExtraSolar.

Se imprime en pantalla la cadena “Boom!”, esto debido al cast explícito que se hace en la línea D (en este caso funciona porque oa apunta a un objeto de la clase Nova), por lo que accede a los métodos propios de la clase Nova.

m) ¿Por qué la línea 15 imprime true? ¿Para cualquier objeto que se cree siempre imprimirá lo mismo? ¿Qué imprimen las siguientes líneas? ¿Por qué? En la línea 15 imprime true porque si bien obN es un apuntador de tipo ObjetoAstronomicoExtraSolar, el objeto al que apunta es de tipo Nova, y toda clase es una subclase de Object en Java. Pasará lo mismo con cualquier otro tipo de objeto.

Ahora que a obN se le asignó null, instanceof Object devuelve false, esto es porque null no es una clase, ni puede generar objetos, es un literal especial.

En cambio la última línea sí retornaría true, porque la suma del valor null con una cadena vacía devuelve la cadena “null”, que es un objeto de la clase String, y por tanto, una instancia subclase de Object.

n. Agregue el siguiente constructor. ¿Se genera error? ¿Se pueden colocar constructores a clases abstractas? ¿Qué sentido tiene? Sí se puede colocar constructores a clases abstractas, es útil para crear un constructor común con los atributes compartidos entre todas las subclases (y todas heredan el método tipoCuerpo2 y el atributo ID), ahora bien, cabe recalcar que la propia clase abstracta no puede ser instanciada con ese ni con ningún constructor.

o. Suponga que agrega la clase EnanaBlanca como se indica a continuación. ¿Qué se puede concluir de esta nueva situación? ¿Qué errores se presentan y cómo podría corregirlos? Si se intenta ejecutar los métodos abstractos que debieron ser definidos (explotar, descripcion), retornará un error por falta de implementación. Además, no podría acceder a su atributo ID. Sería necesario definir métodos abstractos con @Override y agregar métodos get/set para hacer de EnanaBlanca una clase más completa