PROGRAMACIÓN ORIENTADA A OBJETOS - UNALMED 2024 -2

Andrea Merino Mesa

1034991011

1. **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:**

```
public abstract class Instrumento
{
    private String tipo;
    public Instrumento(String tipo) { this.tipo = tipo; }
    public String getTipo()     { return tipo;}

    public abstract void Tocar();
    public abstract void Afinar();
}

public class Saxofon extends Instrumento
{
    public Saxofon(String tipo) { super(tipo); }
    public void Tocar() { System.out.println("Tocando Saxofon");}
    public void Afinar() {System.out.println("Afinando Saxofon");}
}

public class Guitarra extends Instrumento {
    public Guitarra(String tipo) { super(tipo); }
    public void Tocar() { System.out.println("Tocando Guitarra");}
    public void Afinar() {System.out.println("Afinando Guitarra");}
}
```

La clase *Instrumento* debe definirse como abstracta para poder **declarar métodos abstractos, los cuáles no tienen código y deben ser definidos por sus clases hijas**. Además, se diferencia de una clase normal porque obliga a sus hijas a definir los métodos que ella solo declara y también porque **no** permite que se **creen instancias** de ese tipo, aunque define constructor para que los constructores de sus clases hijas lo puedan usar.

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?**

```
public class Test {
    public static void main(String[] args) {
        Instrumento x;
        x = new Saxofon("xxxxx");
        x = new Guitarra("xxxxx");
    }
}
```

Todas las líneas pueden ser ejecutadas, porque la variable x se puede declarar como de tipo *Instrumento*, lo que no se puede hacer es instanciar objetos de tipo *Instrumento* (new()), pero esto no sucede porque a la hora de crear las instancias, estas **se crean apuntando a objetos de las clases hijas**. Además, es válido porque tanto los constructores de las clases Guitarra y Saxofón reciben un String que le pasan al constructor de la clase padre, dónde él se lo asigna al atributo tipo, porque, aunque es privado, como el constructor es de la clase Instrumento y este se modifica dentro del constructor, se puede acceder a él.

d) **¿Qué imprime el siguiente programa?**

```
public class Test {
    public static void main(String[] args) {
        Instrumento x;
        x = new Saxofon("xxxxx");   x.Tocar();
        x = new Guitarra("xxxxx");  x.Tocar();
    }
}
```

Tocando Saxofon //Pues x primero apunta a un objeto de tipo Saxofon.
Tocando Guitarra //Pues x luego pasa a apuntar a un objeto de tipo Guitarra.

2. **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.

```
abstract class Estrella extends ObjetoAstronomicoExtraSolar {
    abstract void explotar() {
        System.out.println("Estrella explotar");
    }
    int a = super.getID();
    public void tipoCuerpol() {
        System.out.println("Simple " + a);
    }
}
```

Sería un **error** porque los métodos declarados como abstractos se declaran, pero no se definen, por tanto, no se pueden usar las llaves ni poner nada adentro {}.

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.

Aunque la clase *ObjetoAstronomicoExtraSolar* es **abstracta, puede tener tanto métodos abstractos como métodos normales.** En este caso, los métodos **tipoCuerpo2() y getID() son métodos normales** pues no se declaran como abstracto. Esta situación **no se considera como errónea** ya que una clase abstracta **puede implementar métodos comunes a todas sus subclases (No abstractos)**, siempre y cuando no los declare como abstractos y a su vez los defina {}.

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.

```
abstract class ObjetoAstronomicoExtraSolar {
    private int ID;

    public void tipoCuerpo2() {
        System.out.println("Extrasolar");
    }

    public int getID() {
        return this.ID;
    }
}
```

No se debería de considerar como un error ya que definir una clase abstracta no solo permite declarar métodos sin definirlos, sino que también permite evitar la creación de instancias de esa clase y a su vez utilizarla únicamente como clase padre. **Por tanto, al no definir métodos abstractos, aún es útil si queremos que no se puedan instanciar objetos de esta clase sino solo de clases hijas.**

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.

Esto sucede puesto que **cada posición (index) del arreglo *oa* se declara como *ObjetoAstronomicoExtraSolar* para que, por polimorfismo, cada posición pueda apuntar a cualquier objeto de una clase hija de la clase abstracta.** Esto es lo que nos permite crear objetos de diferentes clases hijas de *ObjetoAstronomicoExtraSolar* como Galaxia, Nova y SuperNova y por tanto **hacer uso de la ligadura dinámica**, usando referencias de una clase abstracta (ObjetoAstronomicoExtraSolar) hacia objetos de sus subclases (Galaxia, Nova y SuperNova).

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*?

Aunque el objeto al que apuntaba el arreglo oa en esa posición antes era de tipo Galaxia, **en la línea 28, este apuntador pasa a apuntar al mismo objeto que la posición 2 del arreglo, que es un objeto de tipo SuperNova.** Esto puede hacerse ya que todas las posiciones de la lista fueron **declaradas como *ObjetoAstronomicoExtraSolar*** y por tanto **pueden apuntar a cualquier objeto que sea de una clase hija** de este.

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?

A pesar de que la super clase *ObjetoAstronomicoExtraSolar* solicita que todas sus hijas definan los métodos que esta declaró como abstractos, como la clase ***Estrella* es abstracta, no tiene la necesidad de implementar el método abstracto heredado(*descripcion()*).**

g. ¿Qué sucede si el método *tipoCuerpo1()* de la clase Galaxia se define como privado?¿Por qué se genera error?

Aunque la clase Galaxia defina el método *tipoCuerpo1(),* si lo define como privado generará error puesto que **al ser privado ninguna otra clase puede acceder a él,** además, **cuando se sobrescribe un método este debe ser igual o más visible que el de su padre** y el declararlo privado no cumple con ello, resultando en un error en compilación. Esta "ley" se debe a la capacidad de los objetos de ser polimórficos.

h. ¿Por qué la clase *Nova* no define el método *tipoCuerpo1()*? ¿Se podría definir? Si lo define, ¿qué interpreta de esta situación?

Para la clase Nova **no es necesario definir el método *tipoCuerpo1()*** pues esta **hereda de la clase abstracta *Estrella*, la cual ya previamente lo definió.** Sin embargo, se puede definir este método nuevamente en la clase Nova ya que esto contaría como **redefinición de métodos.**

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?

Imprime **la clase a la que pertenece el objeto (Galaxia), seguido de @ y la dirección en memoria del objeto.** Puede llamar al método toString() pues la clase Galaxia hereda de *ObjetoAstronomicoExtraSolar* y este a su vez **hereda de Object,** y por polimorfismo, el apuntador g es a su vez Galaxia, *ObjetoAstronomicoExtraSolar* y Object.

j. ¿Por qué en la línea 11 se puede crear un puntero obN de tipo *ObjetoAstronomicoExtraSolar* si esta es una clase abstracta?

Lo que se hace en la línea 11 es declarar el apuntador obN como *ObjetoAstronomicoExtraSolar* y desde ahí **se apunta a un objeto de tipo Nova**, pero por medio de una conversión de tipos **(generalización)**, ese objeto de tipo nova se **"convierte" a un objeto de tipo *ObjetoAstronomicoExtraSolar***.

k. ¿Las siguientes instrucciones (instrucciones en las líneas B y C) son válidas? Explique.

```
A. Nova nova = new Nova();
B. ObjetoAstronomicoExtraSolar ob = new ObjetoAstronomicoExtraSolar();
C. ObjetoAstronomicoExtraSolar oa = nova;
```

Las instrucciones A y C son válidas pero la B no lo es:

A) **Se puede** crear un objeto de tipo Nova ya que esta es una clase abstracta y el apuntador cumple con ser de igual tipo.

B) **No es correcta** porque no se pueden instanciar objetos de clases abstractas.

C) **Se puede** apuntar desde oa al mismo objeto al que apunta nova puesto que el apuntador de un objeto puede ser de la misma clase o de la superclase como es el caso, sin tener que hacer una conversión de tipos explícita.

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.

```
A. Nova nova = new Nova();
B. ObjetoAstronomicoExtraSolar oa = nova;
C. oa.explotar();
D. ((Nova) oa).explotar();
```

- La instrucción en la línea **B es correcta** porque **se puede** apuntar desde oa al mismo objeto al que apunta nova puesto que **el apuntador de un objeto puede ser de la misma clase o de la superclase** como es el caso, sin embargo, la instrucción en la línea **C es incorrecta** puesto que antes de ejecutar un método por ligadura dinámica, el programa verifica que el tipo del apuntador lo tenga definido, y como ***ObjetoAstronomicoExtraSolar*** **no tiene definido el método explotar (), este genera error.**

- Omitiendo la instrucción C, el programa imprime **Boom!** porque al hacer la especialización de oa, cuando el programa verifique si el tipo del apuntador tiene el método, se irá a la clase Nova en vez de a la clase ***ObjetoAstronomicoExtraSolar.***

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?

```
obN = null;
System.out.println(obN instanceof Object);
System.out.println("" + obN instanceof Object);
```

- La línea 15 **imprime True** porque obN apunta a un objeto que es Nova, que hereda de Estrella, que hereda de *ObjetoAstronomicoExtraSolar y que* hereda de Object, entonces **por polimorfismo obN es a su vez Object.**

- Si, cualquier **objeto que se cree siempre heredará de Object**, por lo que siempre imprime **True.**

- Las siguientes líneas imprimen:

  1) **False**, pues en el primer print obN apunta a **null**, lo cual significa que **aún no apunta a ningún objeto** y por tanto no hereda de object.

  2) En el segundo print lo que pasa es que **"" + obN da como resultado un String**, y como los Strings heredan de Object imprime **True.**

n. Agregue el siguiente constructor. ¿Se genera error? ¿Se pueden colocar constructores a clases abstractas? ¿Qué sentido tiene?

```
public ObjetoAstronomicoExtraSolar() {
    this.ID = 4;
    this.tipoCuerpo2();
}
```

No genera error puesto que **en una clase abstracta puede definirse un constructor** si se desea, lo que no se puede hacer es instanciar objetos de este tipo. El sentido de esta práctica es **definir un constructor para que solamente sea usado por los constructores de sus clases hijas**.

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?

```
class EnanaBlanca extends Estrella {
    void agotarCombustible() {
        System.out.println("Enana blanca muere");
    }
}
```

La clase EnanaBlanca **hereda de Estrella que a su vez hereda de *ObjetoAstronomicoExtraSolar***, por lo que hereda los métodos getID(), tipoCuerpo2(), descripcion(), tipoCuerpo1() y explotar () y el atributo a. Además del método agotarCombustible() que esta misma define.

Los **errores** que se presentan se deben a que la clase EnanaBlanca n**o define los métodos abstractos que hereda, es decir, explotar () y descripcion ().** Este error se podría corregir haciendo que la clase EnanaBlanca sea **abstracta o definiendo estos dos métodos** dentro de esta.