Taller 7 Java ejercicio 2

Jhoneyker Delgado Urbina

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 ya que todos los diferentes tipos de instrumentos deben afinarse y se deben poder tocar, por lo que al definirse como abstracta obliga a todos los que hereden de ella a que redefinan estos métodos dependiendo de su nombre. La principal diferencia es no se encuentra un código definido para los métodos abstractos y no se pueden crear instancias de la clase Instrumento por sí solo.

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

El código definiendo la clase Piano se vería de la siguiente manera:

```
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 del código se pueden ejecutar sin ningún problema.

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();
    }
}
```

El programa imprime lo siguiente:

Tocando Saxofon

Tocando 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);
    }
}
```

Si se define el método explotar de la clase estrella de esa manera al momento de compilar mostrara error, ya que los métodos abstractos están diseñados para no tener un cuerpo para que las clases hijas tengan que dar implementación a esté.

   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.**
   Está situación no podría considerarse como un error ya que se define un cuerpo de código después, por lo que las clases que sean hijas de la clase ObjetoAstronomicoExtraSolar pueden utilizar estos métodos sin redefinirlos ya que los heredaron directamente.

   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, no es un error ya que una clase abstracta también puede tener solo métodos concretos (con implementación) lo cual sirve para generar una base conceptual que no debe ser instanciada directamente.

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 se debe a la generalización, lo que se esta diciendo es que ese arreglo va a contener objetos que sean de tipo ObjetoAstronomicoExtraSolar, pero en las siguientes líneas de código se llena el arreglo con objetos de Nova, SuperNova y Galaxia, y en la línea 25 se invoca el método correspondiente a cada clase derivada debido a que se hace la invocación del método en cada uno de los objetos que se encuentran en el arreglo (Además el método descripción es abstracto por lo que cada clase hija tendrá la obligación de redefinir ese método).

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” ya que en la línea anterior (línea 28) se esta deciendo que oa[0] ya deje de apuntar a lo que estaba apuntando y ahora apunte a lo que apunta oa[2] (La cual está apuntando a un objeto de tipo SuperNova) por lo que el método descripcion() llamado en la línea 29 se ejecutara sobre un objeto de tipo 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?**

En la clase Estrella no es obligatorio definir el método descripcion() ya que esta también es una clase abstracta y ella decide si lo implementa o no, pero ya las clases hijas de esta que no sean abstractas si tendrán la obligación de definirlo.

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

Se genera error porque ese método abstracto no puede tener una accesibilidad menor a la que está definida en la clase padre.

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

Porque Nova hereda de Estrella y está ya creo un cuerpo de código para este método, si se podría definir porque cualquier método se puede sobreescribir teniendo en cuenta algunas reglas. Esta situación demuestra que hay que seguir bien el curso del programa para saber que situación se adapta mejor al caso.

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 imprime: Galaxia@65b54208 que corresponde al tipo de objeto que es y el espacio de memoria donde se encuentra. Es posible llamar al método toString() ya que este es heredado de la clase Object por lo que todos lo tendrán con la misma función si no es sobreescrito en ninguna de las clases.

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

Se puede crear un puntero de tipo ObjetoAstronomicoExtraSolar ya que no hay ninguna restricción para esto, se puede hacer para tener una generalización

más amplia, lo que no se puede es crear una instancia de la clase ObjetoAstronomicoExtraSolar ya que es abstracta.

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;
```

La instrucción en la línea B no es válida ya que no se puede crear una instancia de una clase abstracta y la instrucción en la línea C si es valida ya que ha este proceso se le llama generalización en el que se tiene un apuntador de mayor nivel a un objeto de menor nivel.

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 ya que ha este proceso se le llama generalización en el que se tiene un apuntador de mayor nivel a un objeto de menor nivel y la instrucción en la línea C es incorrecta, ya que, la clase ObjetoAstronomicoExtraSolar no conoce ningún método llamado explotar()(No lo tiene definido). Omitiendo esa instrucción se imprime por pantalla:

Boom!

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);
```

Imprime True porque cualquier objeto que se cree de cualquier clase va a actuar como objeto de la clase Object. Las líneas imprimen:

False

False

Porque no se esta creando ningún objeto, a obN se le esta diciendo que sea nulo por ende no va a pertenercer a la clase Object.

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 se genera error, si se pueden colocar constructores a la clases abstractas y tienen mucho sentido ya que las clases hijas de esta entraran al constructor que se especifique o si no se especifica entrara al constructor que no tiene parámetros.

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");
   }
}
```

En esta situación se puede observar que falta por definirse el método explotar() y descripción() ya que estos dos son abstractos y en ninguna otra parte del código se definieron un cuerpo de código de la que pueda heredar EnanaBlanca. Esto se puede corregir definiendo estos métodos e implementándolos.