PROGRAMACIÓN ORIENTADA A OBJETOS - UNALMED 2024 -2

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");}
}
```

Rta/= La clase Instrumento debe definirse como abstracta, porque cada instrumento tiene comportamientos específicos, en este caso, cómo se toca y cómo se afina, los cuales deben definirse en las subclases específicas. La clase abstracta se diferencia de una clase normal, ya que, esta no puede ser instanciada directamente, además de que puede incluir métodos abstractos, los cuales, no se definen, y deben ser sobrescritos en las subclases.

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

Rta/=

```
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");
    }
}
```

Rta/= Todas las líneas se pueden ejecutar, porque a pesar de que no se puede instanciar una clase abstracta directamente, si se puede declarar una variable de tipo abstracto.

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();
    }
}
```

Rta/= Imprimirá:

Tocando Saxofón

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);
    }
}
```

Rta/= Si se define el método explotar() en la clase abstracta Estrella, cómo se muestra en la imagen, este método deja de ser abstracto, porqué ya se definió. Como resultado, cualquier subclase de Estrella que no redefina el método explotar() utilizará el método definido en la clase abstracta.

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.

Rta/= Significa que estos métodos ya definen el comportamiento del método para todas las subclases, evitando que se deba implementar los métodos por separado. No es un error, ya que estos métodos ofrecen funcionalidad predeterminada y no requieren personalización obligatoria, pero si se pueden sobrescribir.

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;
    }
}
```

Rta/= No es un error, ya que, en Java, una clase abstracta puede tener métodos abstractos, pero no es obligatorio tenerlos.

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.

Rta/= La línea 25 invoca el método descripcion() correspondiente a cada clase derivada gracias a la ligadura dinámica, porque, aunque el arreglo oa está declarado como de tipo abstracto, en tiempo de ejecución se implementa el método descripcion() según el tipo real del objeto almacenado en cada posición del arreglo, imprimiendo el mensaje específico de cada clase.

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?

Rta/= En la línea 29 se imprime "Soy una Super Nova", porque se reasigna oa[0] a un objeto de tipo SuperNova, por lo tanto se implementa el método de dicha clase.

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?

Rta/= Esto se debe a que la clase Estrella también es una clase abstracta, por lo que no está obligada a implementar métodos abstractos de su superclase, esa obligación recae en las subclases que no son abstractas.

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

Rta/= Se genera un error porque para cumplir con la definición de un método abstracto, este debe ser accesible desde la superclase, lo cual no permite que sea privado.

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

Rta/= No es necesario que la clase Nova redefina tipoCuerpo1(), ya que se hereda de la clase Estrella. Si se define en Nova, se sobrescribirá el método heredado e interpretaría el método sobrescrito.

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?

Rta/= En Java, todas las clases heredan de la clase Object, por lo que, no necesitan definir el metodo toString(), ya que lo heredan.

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

Rta/= Porque si puede crear una referencia de tipo abstracto, lo que no se puede hacer es instanciarlo directamente. En este caso, la referencia obN apunta a un objeto concreto de la clase Nova.

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;
```

Rta/= La línea B es no es válida porque una clase abstracta no se puede instanciar y la línea C

es válida porque por polimorfismo un objeto de clase Nova, también es de clase ObjetoAstronomicoExtraSolar.

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();
```

Rta/= La línea C es incorrecta porque el tipo declarado (ObjetoAstronomicoExtraSolar) no tiene acceso directo al método explotar(), y la línea D es correcta porque por especialización a la clase Nova permite llamar al método explotar() de esa clase concreta, lo que haría que se imprimiera el explotar de Nova, “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);
```

Rta/= La línea 15 imprime true porque obN apunta a un objeto que hereda indirectamente de Object. Siempre que la referencia no sea null, instanceof Object devolverá true. Esto se debe a que todas las clases en Java heredan de Object directa o indirectamente. La primera línea imprime false porque la referencia obN es null y la segunda no compila por un error de escritura.

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();
}
```

Rta/= No, no se genera un error y las clases abstractas pueden tener constructores, las subclases pueden invocar estos constructores mediante la palabra clave super(). Tener constructores en una clase abstracta sirven para inicializar atributos comunes entre todas las subclases.

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");
    }
}
```

Rta/= Lo que se puede concluir es que la clase EnanaBlanca no es abstracta, por lo que está obligada a implementar todos los métodos abstractos heredados de la clase Estrella y de sus superclases. Como descripcion() es abstracto en ObjetoAstronomicoExtraSolar y no ha sido implementado en Estrella, debe ser definido en EnanaBlanca. Como no lo hace, se genera error al compilar. Para corregirlo se debe implementar el método.