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 ser abstracta al tener métodos abstractos, sino habría error, es decir, las clases normales no pueden tener métodos abstractos, por otro lado, a diferencia de las clases normales, no se puede crear objetos de las clases abstractas.

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");
    }
}
```

De hecho yo creo que si ejecuta ya que x seria un apuntador de clase abstracta instrumento, más no se está intentando crear un objeto de clase instrumento como tal, además, como saxofón y guitarra son sus clases “hijas” entonces seria como una conversión implícita (generalización), donde el apuntador x quedaría señalando un objeto de clase Guitarra.

d) ¿Qué imprime el siguiente programa?

Imprime lo siguiente ya que x primero apunta a un objeto de clase Saxofon y luego apunto a un objeto de clase Guitarra.

Tocando Saxofon

Tocando Guitarra

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

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.

Habría un error de compilación ya que es un método abstracto y estos se caracterizan por declararse pero sin ningún código en su "interior" o cuerpo.

```
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);
    }
}
```

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: No seria un error porque las clases abstractas pueden tener métodos normales o concretos, y se heredarían normalmente a las clases hijas.

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.

Las clases abstractas se pueden definir sin métodos abstractos, ya que estos son una posibilidad más no una obligación, es decir, si una clase tiene métodos abstractos entonces es abstracta, pero si un clase es abstracta no necesariamente debe tener métodos abstractos.

```
abstract class ObjetoAstronomicoExtraSolar {
    private int ID;

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

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

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 19 hace referencia a que la lista que admite objetos de clase ObjetoAstronomicoExtraSolar, aun si esta es abstracta y no puede ser instanciada, ya que esto implica que admitiría objetos heredados de esta clase, por el principio de polimorfismo, de esta forma en la línea 25, que está dentro de un ciclo for que recorre los elementos de la lista, se ejecuta el método descripción() dependiendo a que clase pertenecen los objetos, ya sea Galaxia, Nova o 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*?

Debido a la línea 28, donde la referencia al objeto en la posición 0 de la lista *oa* queda apuntando al objeto en la posición 2, es decir el objeto de clase Supernova, así que en la siguiente línea se ejecuta el método descripción () de esta clase.

```
oa[0] = oa[2];
oa[0].descripcion();
```

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 es abstracto?

Porque Estrella también es una clase abstracta, así que no necesita definir todos los métodos abstractos de la clase "padre", que en este caso es ObjetoAstronomicoExtraSolar.

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

Porque en la clase ObjTaller7 se llama al método *tipoCuerpo1()* y este al volverse privado sólo se conoce en la propia clase en la que está definido, pudiendo ser accedido tan sólo en ella.

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

Porque la clase abstracta Estrella ya la definió, así que realmente Nova heredo esta definición, aunque también se podría sobreescribir si fuera necesario. Esto quiere decir que el método *tipoCuerpo1()* puede ser llamado desde un objeto de clase Nova. (ej:nova1.*tipoCuerpo1())*

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 Galaxia@591f989e.
Y se puede llamar el método toString() desde cualquier clase ya que todas las clases en Java (en este caso Galaxia y *ObjetoAstronomicoExtraSolar*) son heredadas de Object de una forma u otra, así que se le considera una superclase.

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

Porque una referencia de tipo abstracto, en este caso *ObjetoAstronomicoExtraSolar*, simplemente puede apuntar a objetos de las clases "normales" que heredan de la clase abstracta, incluyendo la clase Nova, a la cual pertenece el apuntador nova.

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

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

La línea B no seria valida ya que esta intentando instanciar una clase abstracta, es decir, crear un objeto de clase *ObjetoAstronomicoExtraSolar,* por otro lado la línea C no generaría error puesto que el apuntador oa de tipo abstracto apuntaría al objeto con referencia nova, es decir un objeto de clase Nova, un caso de generalización valido.

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

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.

Como se dijo en el punto anterior la línea B sería una generalización valida, pero la C es incorrecta ya que para que se pueda ejecutar debe cumplir unas condiciones entre ellas que la clase del apuntador al menos tenga definido el método, el cual no es el caso, *ObjetoAstronomicoExtraSolar* no tiene el método *explotar().*

Por otro lado, ignorando la línea C se imprimiría: "Boom!", ya que seria una conversión explicita en la línea D para que se pueda ejecutar el método explotar(), esta conversión tiene lugar gracias a que oa realmente apunta a un objeto de clase Nova.

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

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?

Porque todo objeto en Java es de clase Object por polimorfismo, y como obN apunta a un objeto de clase Nova entonces el instanceof retorna True.
Por otro lado yo supongo que en las siguientes líneas se imprimiría: False, puesto que como obN no está apuntando realmente un objeto, debido al null, entonces el instanceof debe retornar falso ya que no hay un objeto al cual verificarle la pertenencia a un clase

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

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

No se genera error, ya que, si se le pueden poner constructores a clases abstractas, solo que no se usaran para instanciar o crear objetos de esta clase *ObjetoAstronomicoExtraSolar*, sino que puede ser llamado por los constructores de las clases “hijas” cuando estas estén siendo creadas.

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

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?

El principal error que se debe corregir es la falta de los métodos explotar() y descripción(), puesto que estos son los métodos abstractos definidos en las clases Estrella y *ObjetoAstronomicoExtraSolar* respectivamente.

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

PROGRAMACIÓN ORIENTADA A OBJETOS - UNALMED 2024 -2