Taller 7 Java, punto #2

1.Ejercicio 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");}
}
```

- Debido a que la clase Instrumento tiene métodos abstractos, es necesario definirla como una clase abstracta (o no compilará el código)
Se diferencia de una clase normal en que no puede ser iterada en un objeto (Aunque naturalmente, todo objeto hijo iteración de una clase hija de la clase abstracta, tiene un apuntador de la clase abstracta de la que es hijo), y su propósito real es servir como plantilla para la creación de clases hijas, así como un apuntador general para la mismas.

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

```
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 puede ejecutar y por qué?

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

- Ninguna línea se puede ejecutar, esto debido a que se están definiendo las clases Saxofon, Instrumento y Guitarra en el mismo archivo Java, con “public”.
En Java, solo se puede tener una clase pública por archivo.
- Si consideramos que las clases están en efecto en sus respectivos archivos

Instrumento.java, Saxofon.java y Guitarra.java, entonces no existe ningún problema y las líneas se ejecutan normalmente.

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();
   }
}
```

- Imprime las líneas, “Tocando saxofón”, “Tocando Guitarra”.

Debido a que x se trató como un apuntador de tipo Instrumento, no existe ningún problema al asociarla con cualquier objeto de una clase más específica que su apuntador.
Por lo anterior, y debido a que en si redefinimos un método en Java, dentro de una clase específica, siempre que se llame al método, sin importar si el apuntador desde que se llama al método del objeto no es de la misma especificidad del objeto en cuestión, se utiliza el método de la clase más específica.

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

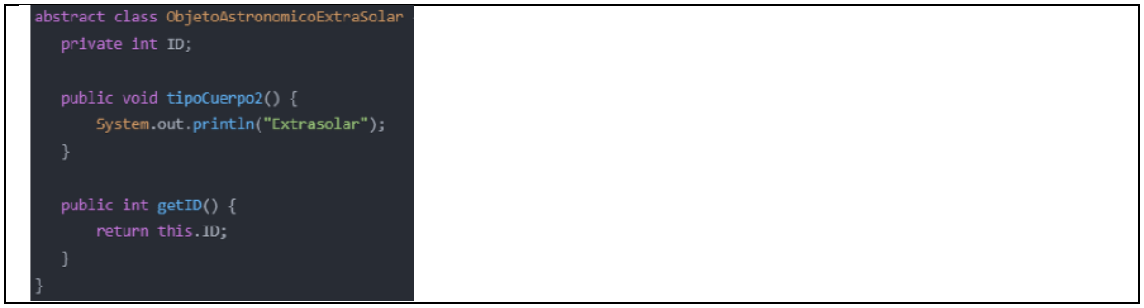
```
abstract class Estrella extends ObjetoAstronomicoExtraSolar {
   abstract void explotar() {
       System.out.println("Estrella explotar");
   }
   int a = super.getID();
   public void tipoCuerpol() {
       System.out.println("Simple " + a);
   }
}
```

- El código no compila, debido a que los métodos abstractos no pueden tener cuerpo.
No tiene sentido definir un método abstracto y darle un cuerpo, pues el propósito de este es simplemente servir como una plantilla, y si se desea crear un cuerpo al método, entonces este no puede ser abstracto, o habrá un error de compilación

b) ¿Qué significa que los métodos tipoCuerpo2() y getID() de la clase *ObjetoAstronomicoExtraSolar*, no se definan como *abstract*? ¿Podría considerarse esta situación como un error? Explique
- No es un error de compilación, pero podría considerarse un error conceptual humano, dado que entendemos una clase abstracta como una plantilla que por si misma no define el contenido de los métodos o atributos, si no solo define que existen los mismos.
Sin embargo, dado que es posible crear métodos no abstractos en una clase abstracta, y que estos sean heredados, utilizarlo no necesariamente es un error si el método no

abstracto que creamos es uno que queremos que sea igual (o si no, ser redefinido en su respectiva clase) y aparezca en todas las clases hijas de la clase abstracta, ahorrando el problema de escribirlo para cada una, o crear una clase no abstracta, hija de la abstracta, que contenga este.

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;
    }
}
```

- Aunque conceptualmente sí, dado que no genera errores de compilación por sí misma, no en todos los casos.
Sea el caso en que queremos crear un apuntador general para todo un tipo de clases que creemos, podemos utilizar esto, creando una clase abstracta que contenga métodos no abstractos que serán compartidos por todas las clases hijas, pero que no tiene ningún método abstracto, para utilizarla exclusivamente como apuntador para generalizar todas nuestras clases hijas creadas, y que esta clase abstracta no pueda ser inicializada en algún objeto.

d) Explique por qué el arreglo oa (línea 19) hace referencia a una clase abstracta y sin embargo, en la línea 25 se invoca al método correspondiente a cada clase derivada.

- Debido a que en java, cuando redefinimos un método del padre en una clase hija, sin importar el apuntador desde el que estemos llamando al método del objeto, si el objeto es una clase hija del apuntador, entonces se llama al método más específico.
Esto es igual incluso si el apuntador es una clase abuelo, o superior, a la clase del objeto al que está apuntando.

e) ¿Por qué en 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 que se cambió el objeto al que el apuntador, que es de tipo *ObjetoAstronomicoExtraSolar* se cambió de apuntar al objeto inicializado Galaxia, a un objeto de tipo SuperNova.

f) ¿Por qué en la clase *Estrella* no se define el método *descripción()* si la superclase lo está solicitando, ya que el método es abstracto?

- Debido a que la clase Estrella también es abstracta, y por tanto, no es necesario utilizar o definir todos los métodos abstractos de la clase padre.
Esto es una característica de Java (y Python).

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

- Debido a que el método tipoCuerpo1() de la clase padre tiene un nivel de accesibilidad mayor, o menos encapsulado, que privado.
En java, no se puede redefinir el nivel de accesibilidad de un método que es heredado, a un nivel menor al de la clase padre. Lo contrario si es posible (Redefinir un método privado como public)

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

- No lo define debido a que es una clase "nieta" de la clase ObjetoAstronomicoExtraSolar, y su clase padre, que es Estrella, ya lo ha definido, efectivamente heredando el método definido a la clase Nova.
Si podría ser definido, en este caso, sería una redefinición respecto al de su clase padre, Estrella.

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@407a7f2a", que un identificador del objeto.
Debido a que ambos son hijos de una clase "Object", que tiene el método toString() definido y hereda a todos los objetos en Java.
Object es una clase propia de java, que es padre de todos los objetos (O abuelo, etc)

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

- No hay ningún problema con crear un puntero de una clase, sin importar si es abstracta, dado que lo que no se puede con las clases abstractas es crear una instancia, pero un apuntador no es una instancia, por tanto, es posible y no existe ningún problema al hacerlo.
Este puntero, naturalmente, solo puede apuntar a los objetos hijos de la clase abstracta en cuestión, pues todo objeto de una clase hija es un objeto de la clase en la que se inicializó, y de todas las clases de las que haya heredado esta. (Pues nunca va a existir un objeto que sea solamente de la clase abstracta del apuntador)

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;
```

- No, debido a que se está intentando crear una instancia de objeto de una clase abstracta, cosa que no es posible en Java.
Si solo se estuviese creando el apuntador en la línea B (Sin la asignación), entonces sería válido y el resto de las instrucciones no generarían ningún error de compilación.

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();
```

- Debido a que no existe el método explotar para la clase ObjetoAstronomicoExtraSolar. Este método aparece en la clase Estrella, que es hija de ObjetoAstronomicoExtraSolar, y que luego es heredado por Nova.
Dado que el apuntador de oa es de tipo ObjetoAstronomicoExtraSolar, solo se pueden llamar los métodos definidos para esta clase desde este apuntador, y en caso de se llamen, luego se utiliza el método más específico, pero en primer lugar este método tiene que existir para la clase desde la que se está apuntando.
La línea D es correcta dado que estamos usando un apuntador Nova, que si tiene el método explotar() definido.
- Imprime "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 debido a que obN apunta a nova, y este objeto nova es de la clase Nova, que es hija de la clase Estrella, que es hija de la clase ObjetoAstronomicoExtraSolar, que es hija de la clase Object, y por tanto, el objeto nova es de las clases Nova, Estrella, ObjetoAstronomicoExtraSolar y Object, todo a la vez.

- Sí, imprimirá lo mismo siempre, dado que todo objeto es una clase hija, o nieta, o cualquier otro nivel de herencia, de la clase Object. (Siempre que sea un objeto, un dato primitivo naturalmente no imprimirá lo mismo)

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

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

- No genera ningún tipo de error de compilación, significando que si se pueden crear constructores en una clase abstracta.
Tiene sentido, pues aunque la clase en si misma no puede tener iteraciones de objetos, el constructor es una característica de toda clase, y por tanto, las clases abstractas pueden definir constructores aunque no sean utilizados.
Conceptualmente, crear un constructor en una clase abstracta puede ser interpretado como crear un constructor general que será utilizado (O puede ser utilizado) por todas las clases hijas de la clase abstracta.

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 como podría corregirlos?

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

- Lo anterior presenta errores debido a que no se están definiendo todos los métodos abstractos de la clase Estrella y la clase ObjetoAstronomicoExtraSolar (Con excepción de tipocuerpo1(), ya que este método es definido en Estrella)
- Para corregirlos basta con definir estos métodos.

```
class EnanaBlanca extends Estrella{
  void agotarCombustible(){
    System.out.println("Enana blanca muere");
  }
  void explotar(){
    System.out.println("MEGA-BOOM!!");
  }
  void descripcion(){
    System.out.println("*Se describe*");
  }
}
```

Lo anterior no presenta ningún problema de compilación:

```
EnanaBlanca boa = new EnanaBlanca();
boa.explotar();
boa.descripcion();
boa.agotarCombustible();
```

```
MEGA-BOOM!!
*Se describe*
Enana blanca muere
User program finished
```