EJERCICIO 2 TALLER 7 JAVA-PYTHON (CLASES ABSTRACTAS)

Tomas Aristizabal Gomez - 1033181207

1. **Ejercicio Análisis**
    a. La clase **Instrumento** debe definirse como abstracta para representar un concepto general, que no debe ser una instancia, en este caso por ejemplo no tiene sentido tener un “Instrumento genérico” sino que debe ser extendida a clases más específicas. No puede ser instanciada directamente; obliga a diferencia de clases normales, a cada una de sus subclase a proporcionar su propia implementación de un método “común” entre ellas… Además se utiliza como base para otras clases
    b.
```
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. Todo se puede ejecutar porque no se está creando una instancia de **Instrumento**, solo una referencia
    d. Imprime: “**Tocando Saxofon**
    Tocando Guitarra”

2. **Ejercicio de código en el repositorio**
    a. Generaría un error, debido a que los métodos abstractos no pueden contener un cuerpo ni realizar acciones.
    b. No es un error, son métodos normales con una implementación, estos sí pueden ser usados por las subclases como están definidos y pueden ser sobrescritos
    c. No, no es un error ya que las clases abstractas pueden o no tener métodos abstractos, todo depende del contexto del programa, tal vez para evitar que hayan instancia de **ObjetoAstronomicoExtraSolar** o para agrupar lógica común…
    d. El arreglo **oa** está diseñado específicamente para tener objetos de tipo **ObjetoAstronomicoExtraSolar** es necesario recordar que todas sus subclases como **Galaxia, Nova, SuperNova** son también del tipo **ObjetoAstro…** entonces cuando se hace el llamado a cada uno de los elementos de este arreglo se invoca al método **descripcion()** especifico de cada subclase
    e. Porque en la línea anterior: `oa[0] = oa[2];` entonces oa[0] dejó de apuntar a **Galaxia** y empezó a apuntar a el que apuntaba oa[2] osea una **Nova**
    f. Porque la clase **Estrella** es también de tipo abstracto y hereda de esta, por lo cual no es necesario definir nuevamente el método **descripcion()**
    g. Generaría un error debido a que no se puede disminuir la visibilidad del método abstracto definido en la clase padre abstracta, Java exige que la visibilidad sea igual o más amplia para evitar conflictos con la jerarquía

h. La clase **Nova** no define este método debido a que su clase padre **Estrella** que hereda de **ObjetoAstronomicoExtraSolar** donde esta definido el método abstracto originalmente, lo define nuevamente de manera normal (no abstracta) por ende es heredado a **Nova** de manera normal y puede hacer uso de la definición que se le dió a este en **Estrella** sin necesidad de redefinirlo. Si se define nuevamente estaría ocurriendo una sobrescritura como ya la conocemos. Se puede interpretar y entender el hecho de que clases abstractas que heredan de otras abstractas pueden definir métodos abstractos de sus clases padres como métodos normales para generalizarlo en todas sus hijas y que también pueden ser sobreescritos
i. En este caso se está llamando al método **toString** de la clase **Object** esto ocurre cuando el método no está definido lo que va a imprimir es una representación en cadena de la dirección en memoria del objeto, algo así: **Galaxia@19469ea2**
j. El puntero **obN** puede ser de tipo **ObjetoAstronomicoExtraSolar** porque **Nova**, como subclase concreta de **ObjetoAstronomicoExtraSolar**, cumple con la herencia y permite que **nova** sea tratado como un objeto de tipo padre **ObjetoAstronomicoExtraSolar**
k. La línea **B** no es correcta debido a que se está intentando instanciar una clase abstracta. En la línea **C** si se puede hacer debido a la relación de herencia entre **Nova** y **ObjetoAstronomicoExtraSolar** lo que significa que **nova** (instancia de Nova) puede ser referenciada por un puntero de tipo **ObjetoAstronomicoExtraSolar**
l. La línea **B** es correcta debido a la relación de herencia entre **Nova** y **ObjetoAstronomicoExtraSolar** lo que significa que **nova** (instancia de Nova) puede ser referenciada por un puntero de tipo **ObjetoAstronomicoExtraSolar**; La línea **C** es incorrecta debido a que el método explotar no está definido en la clase **ObjetoAstronomicoExtraSolar** ya que Java se basa en el tipo estático de la referencia para verificar los métodos accesibles. Omitiendo la línea **C** se imprime por pantalla: **“Boom”** ya que está habiendo un cast explícito a la referencia para tratar a **oa** como un objeto de tipo **Nova** temporalmente, lo que permite acceder a los métodos definido en **Nova** como **explotar()**
m. La línea 15 imprime **true** y siempre lo hará debido a que la clase **Object** es la clase padre de todas las otras clases, por ende cualquier instancia de cualquier objeto es también instancia de la clase **Object.** La primera parte imprime **“false”** debido a que **null** no es una instancia de ninguna clase. La segunda parte imprime “**true”** debido a que al agregar **“” +** ocurre una concatenación con el valor de **obN** dando por resultado un **String: “null”** y **String** hereda de **Object** por lo cual la salida es **true**
n. El nuevo constructor no generaría error, ayudaría a inicializar atributos comunes y realizar configuraciones básicas para las subclases
o. Para que la nueva clase **EnanaBlanca** funcione correctamente debe definir los métodos faltantes abstractos de su clase padre, en este caso **descripcion()** y

explotar(). Se puede concluir que **EnanaBlanca** es de tipo **Estrella** y al mismo tiempo de tipo **ObjetoAstronomicoExtraSolar**