EJERCICIO 2 TALLER 7 JAVA-PYTHON.

1.a) Explique por qué la clase Instrumento debe definirse como abstracta y qué la diferencia de una clase normal:

La clase Instrumento actúa como una “plantilla general” que no se puede instanciar directamente, pues no tiene sentido usar un instrumento genérico sin implementar acciones específicas como Tocar y Afinar. Al declararla abstracta, las subclases (Saxofon y Guitarra) deben proporcionar sus propias implementaciones de estos métodos, mientras que atributos y métodos comunes como tipo y getTipo se heredan para evitar duplicación de código. A diferencia de una clase normal, una clase abstracta no puede ser instanciada y permite definir métodos abstractos que las subclases deben implementar, promoviendo un diseño más flexible y reutilizable.

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

```java
public class Piano extends Instrumento {
    public Piano(String tipo) {
        super(tipo);
    }

    @Override
    public void Tocar() {
        System.out.println("Tocando el piano");
    }

    @Override
    public void Afinar() {
        System.out.println("Afinando el piano");
    }
}
```

1.c) ¿Señale cuál de las siguientes líneas no se pueden ejecutar y por qué?:

El código es válido (teniendo en cuenta la secuencia de los códigos anteriores) y todas las líneas pueden ejecutarse correctamente. La variable x es de tipo Instrumento, que es una clase abstracta, y se le asignan objetos de las subclases concretas Saxofon y Guitarra. Esto es posible porque las subclases implementan los métodos abstractos de Instrumento. Sin embargo, si alguna de las subclases no implementa correctamente los métodos abstractos Tocar o Afinar, se generaría un error de compilación. Además, si se intentara crear una instancia directa de Instrumento, también provocaría un error, pero en este caso no sucede.

1.d) ¿Qué imprime el siguiente programa?:

El programa imprime "Tocando Saxofon" y "Tocando Guitarra". Primero, se crea un objeto Saxofon y se llama a su método Tocar(), que imprime el mensaje correspondiente. Luego, se crea un objeto Guitarra y se llama a su método Tocar(), lo que imprime el mensaje de la guitarra. Esto ocurre debido al polimorfismo, donde la variable x de tipo Instrumento referencia diferentes clases concretas.

2.a) ¿Qué sucede si se define el método explotar() de la clase Estrella como se indica a continuación? Explique su respuesta:

Si se define el método explotar() en la clase Estrella con una implementación, como se muestra en el código, ocurriría un error de compilación. Esto es porque explotar() está declarado como abstracto en la clase Estrella, lo que implica que no debe tener implementación. Un método abstracto debe ser solo una declaración sin cuerpo, y no se puede definir una implementación dentro de una clase abstracta, ya que se espera que las subclases concreten este método. Si se quiere proporcionar una implementación, el método no debe ser abstracto.

2.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:

Que los métodos tipoCuerpo2() y getID() en la clase ObjetoAstronomicoExtraSolar no se definan como abstractos significa que tienen una implementación concreta en la clase base. Esto es válido si se quiere proporcionar un comportamiento común para todas las subclases de ObjetoAstronomicoExtraSolar, lo que evitaría que cada subclase tenga que implementar estos métodos. No es un error, ya que no todos los métodos en una clase abstracta necesitan ser abstractos; algunos pueden tener implementación para ser reutilizados por las subclases. Sin embargo, si se espera que las subclases modifiquen o sobrescriban estos métodos, sería mejor marcarlos como abstractos.

2.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:

No, no se considera un error definir una clase abstracta sin métodos abstractos, es simplemente una elección de diseño. Una clase abstracta puede contener tanto métodos abstractos como concretos. En este caso, aunque la clase ObjetoAstronomicoExtraSolar no tenga métodos abstractos, sigue siendo abstracta porque podría ser una clase base para otras clases que necesiten heredar de ella y agregar sus propios métodos abstractos o concretos.

2.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:

El arreglo oa hace referencia a una clase abstracta (ObjetoAstronomicoExtraSolar), pero esto no impide que se puedan almacenar objetos de sus clases derivadas (como Galaxia, Nova y SuperNova). La razón por la que se invoca el método correspondiente a cada clase derivada en la línea 25 es por el polimorfismo. Aunque el arreglo sea de tipo ObjetoAstronomicoExtraSolar, cada elemento del arreglo es una instancia de una subclase concreta, por lo que cuando se llama a descripcion(), Java invoca la implementación específica de ese método para cada tipo de objeto (como Galaxia, Nova o SuperNova). Esto es posible porque el método descripcion() está sobrescrito en las subclases y el tipo real del objeto en cada índice determina qué implementación del método se ejecuta.

2.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?:

Porque, aunque el arreglo oa fue inicialmente llenado con un objeto de tipo Galaxia, en la línea anterior (28) el objeto en la posición 0 del arreglo fue reemplazado por una instancia de la clase SuperNova. Este cambio provoca que cuando se llama a oa[0].descripcion(); en la 29, se ejecute el método sobrescrito descripcion() de la clase SuperNova, que imprime "Soy una Super Nova".

2.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 se define el método descripcion() porque, aunque la superclase ObjetoAstronomicoExtraSolar lo haya definido como abstracto, la clase Estrella también es abstracta. Las clases abstractas no están obligadas a implementar métodos abstractos de su superclase; simplemente deben garantizar que las subclases concretas implementen esos métodos. En este caso, Estrella actúa como una clase intermedia que puede dejar la implementación del método descripcion() a sus subclases concretas, como Nova y SuperNova.

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

Si el método tipoCuerpo1() de la clase Galaxia se define como privado, se generaría un error al intentar invocar este método desde fuera de la clase, como en el caso de nova.tipoCuerpo1() en la clase ObjTaller7. El error ocurre porque los métodos privados solo son accesibles dentro de la misma clase en la que se definen, y en este caso, se intenta acceder a él desde otra clase (por ejemplo, desde Nova).

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

La clase Nova no define el método tipoCuerpo1() porque hereda este método de la clase Estrella, que a su vez lo hereda de ObjetoAstronomicoExtraSolar. En este caso, no es necesario que Nova defina el método, ya que lo hereda de sus clases superiores. Sin embargo, sí se podría definir el método tipoCuerpo1() en la clase Nova si se quisiera sobrescribir la implementación heredada. Si Nova lo define, se interpretaría como una sobrescritura del método de la clase Estrella, lo que indicaría que Nova proporciona una implementación personalizada de tipoCuerpo1(), en lugar de utilizar la versión heredada de Estrella.

2.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 System.out.println(g.toString()); imprime la representación en cadena del objeto g de tipo Galaxia. Aunque la clase Galaxia y su superclase ObjetoAstronomicoExtraSolar no definen explícitamente el método toString(), Java utiliza una implementación heredada desde la clase Object. Todas las clases en Java heredan la implementación de toString() desde Object, que devuelve una cadena que incluye el nombre de la clase y su hash. Así, aunque Galaxia no lo defina, al ser una subclase, utiliza la versión heredada desde Object.

2.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 porque no se está creando una instancia directa de la clase abstracta, sino que se está asignando una instancia de una clase concreta (Nova) a una variable de tipo ObjetoAstronomicoExtraSolar. Las clases abstractas no pueden ser instanciadas directamente, pero pueden ser referenciadas a través de sus subclases concretas. En este caso, nova es un objeto de la clase Nova, que es una subclase concreta de ObjetoAstronomicoExtraSolar, por lo que la asignación es válida y se puede realizar sin problema.

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

La instrucción B no es válida porque ObjetoAstronomicoExtraSolar es una clase abstracta, y en Java no se pueden crear instancias de clases abstractas. Solo se pueden crear instancias de clases concretas que extiendan de una clase abstracta. Por el contrario, la instrucción C es válida, ya que nova es un objeto de la clase Nova, que es una subclase concreta de ObjetoAstronomicoExtraSolar y se realiza una asignación

de un objeto de tipo Nova a una variable de tipo ObjetoAstronomicoExtraSolar, lo cual es posible debido al polimorfismo en Java.

2.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:

La instrucción en la línea B es correcta porque nova es un objeto de la clase Nova, que es una subclase de ObjetoAstronomicoExtraSolar. En Java, un objeto de una subclase puede ser asignado a una variable de tipo de su superclase, lo que permite el uso de polimorfismo. Así, la asignación es válida, y oa ahora hace referencia a un objeto Nova; por otro lado, la instrucción en la línea C es incorrecta porque oa es de tipo ObjetoAstronomicoExtraSolar, que no tiene definido el método explotar(). Aunque oa haga referencia a un objeto de tipo Nova, la variable está declarada como ObjetoAstronomicoExtraSolar, y el compilador no sabe que el objeto es realmente una instancia de Nova sin realizar un cast explícito.

Omitiendo la instrucción en la línea C, al llamar a explotar(), se ejecuta la implementación de Nova, que imprime "Boom!", ya que es el comportamiento definido en esa clase. Esto ocurre porque, en la línea D, se realiza un casting de la variable oa a tipo Nova mediante ((Nova) oa), lo que permite acceder al método explotar() de la clase Nova.

2.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?:

- La línea 15 imprime true porque en Java, todas las clases, ya sean abstractas o concretas, heredan de la clase Object, que es la raíz de la jerarquía de clases. Por lo tanto, cualquier objeto, incluida una instancia de Nova o cualquier otro tipo, siempre será considerado una instancia de Object, lo que hace que la expresión obN instanceof Object sea verdadera.

- No, no siempre imprimirá lo mismo. Si obN es null, la comprobación instanceof devuelve false porque null no es una instancia de ningún tipo, ni de Object.

- La línea obN = null; System.out.println(obN instanceof Object); imprimirá false (porque obN es null, y null no es una instancia de ninguna clase). La línea System.out.println(" " + obN instanceof Object); imprimirá false (porque el operador + se usa para concatenar el string vacío con el valor de obN instanceof Object, lo que evalúa primero la expresión obN instanceof Object, que resulta en false; luego, se convierte false a un string y se concatena con el espacio, resultando en "false", que se imprime en pantalla).

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

No se genera error al agregar el constructor en la clase abstracta. Las clases abstractas pueden tener constructores, y en este caso, el constructor se utiliza para inicializar atributos comunes y ejecutar métodos comunes como tipoCuerpo2(). Aunque no se puede crear una instancia de la clase abstracta directamente, este constructor sirve para preparar la inicialización antes de que las subclases agreguen su propia lógica.

2.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?:

La clase EnanaBlanca extiende Estrella, lo que la convierte en una subclase de una clase abstracta, y se espera que implemente los métodos abstractos de Estrella. Sin embargo, en la clase EnanaBlanca no se ha implementado el método explotar(), que es abstracto en la clase Estrella, lo que genera un error de compilación. Para corregir esto, se debe agregar la implementación del método explotar() en EnanaBlanca, ya que es obligatorio implementar los métodos abstractos heredados de clases abstractas. La corrección sería algo como:

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

    @Override
    void explotar() {
        System.out.println(x:"Explotando enana blanca");
    }
}
```