

Solución taller 7.

Programación orientada a objetos. Grupo 1

Alejandro Pérez Barrera, alperezba@unal.edu.co, C.C. 1023629729

1.a) Instrumento debe de definirse como abstracta, primero porque cuenta con métodos abstractos, lo cual obliga a Instrumento a ser abstracta, segundo, porque Instrumento se utiliza para darle propiedades comunes a sus subclases (Saxofón y Guitarra), y tercero, porque no se puede crear un objeto de tipo *Instrumento* sin más, es necesario especificar exactamente, cuál instrumento se desea crear. Lo que distingue de una clase normal es justamente que, ni en el código, ni en la vida real, es posible la creación o la existencia de **El Instrumento**, hace falta nombrar a un instrumento en particular.

1.b)

```
J Piano.java > ...
1  public class Piano extends Instrumento{
2
3      public Piano(String tipo){
4          super(tipo);
5      }
6
7      public void Tocar(){
8          System.out.println(x:"Tocando Piano");
9      }
10
11     public void Afinar(){
12         System.out.println(x:"Afinando Piano");
13     }
14
15 }
16
```

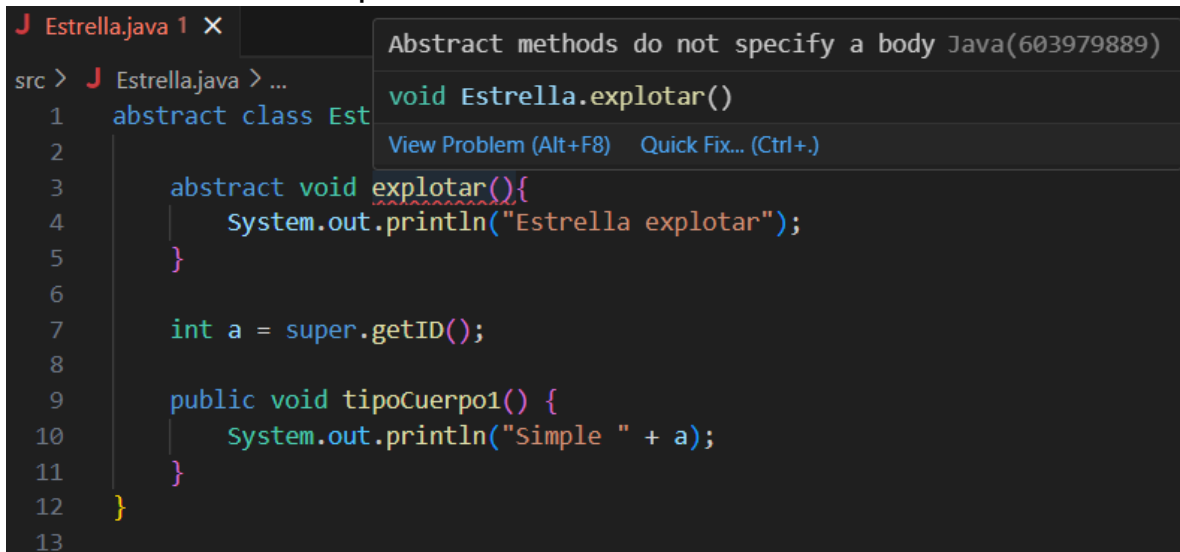
1.c) Técnicamente hablando, todas las líneas se pueden ejecutar porque *Instrumento* *x* **no** está creando ningún objeto, luego *x=new Saxofón("xxxxx");* crea un objeto tipo Saxofón y *x=new Guitarra("xxxxx");* crea un objeto de tipo Guitarra, ambos completamente válidos. De cierta manera, los objetos de tipo Guitarra y Saxofón son simultáneamente objetos de tipo Instrumento.

1.d)

```
Tocando Saxofón
Tocando Guitarra
```

Porque primero *x* apunta a un objeto tipo Saxofón, luego pasa a apuntar a un objeto tipo Guitarra.

2.a) Inicialmente se muestra el error en el editor de código, ya que un método abstracto no puede ser definido



```
src > J Estrella.java > ...
1  abstract class Est
2
3      abstract void explotar(){
4          System.out.println("Estrella explotar");
5      }
6
7      int a = super.getID();
8
9      public void tipoCuerpo1() {
10         System.out.println("Simple " + a);
11     }
12 }
13
```

Abstract methods do not specify a body Java(603979889)

void Estrella.explotar()

View Problem (Alt+F8) Quick Fix... (Ctrl+.)

Sin embargo, el código se ejecuta de la misma manera a previa la modificación

```
Simple 0
Boom!
Boom!
Compuesto
```

Sin embargo, y guiándose únicamente por la teoría, esta modificación es incorrecta, precisamente porque un método abstracto no puede especificar un cuerpo.

2.b) No es un error. Significa que estos métodos se implementan de una manera específica, común para todas las subclases de *ObjetoAstronomicoExtraSolar*, a no ser que alguna los sobre escriba y los implemente de otra forma. No es obligatorio que una clase abstracta tenga únicamente objetos abstractos.

2.c) No, no está mal definir una clase abstracta sin métodos abstractos, lo que hace el método abstracto es el requerirle a una subclase, no abstracta, a que implemente el método a su propia manera. Si una clase abstracta no tiene métodos abstractos, aun así, le puede seguir suministrando a sus subclases comportamientos comunes entre sí.

2.d) Es debido al polimorfismo. Los objetos de tipo Galaxia, Nova y Supernova son simultáneamente objetos de tipo *ObjetoAstronomicoExtraSolar*, lo cual significa que pueden existir dentro del arreglo creado en la línea 19, y contar con un método descripción que sea plenamente funcional, ambas al mismo tiempo.

2.e) Porque en la línea 28, la posición 0 del arreglo oa, pasa a apuntar al elemento de posición 2, por lo que pasa de apuntar a una Galaxia (Antiguo elemento en posición 0), a apuntar a una Supernova (Elemento actual en posición 2).

2.f) Porque Estrella también es un elemento de tipo abstracto, lo cual significa que no está obligada a definir el método de su superclase. Es como si Estrella le pasase la papa caliente de definir a descripción a sus subclases, Nova y Supernova, las cuales si se ven en la obligación

de definir este método porque ninguna de las dos es una clase abstracta.

2.g) Se genera error porque una subclase no puede hacer de un método heredado más privado de lo que ya es, es decir, un método sin modificador de acceso no se puede hacer `private`.

2.h) Nova no lo define porque ya Estrella lo hizo, y hereda precisamente esa definición de su superclase, sin embargo, si Nova ha de implementar el mismo método, lo puede sobre escribir, tales que un objeto de tipo Nova que llame al método, ejecutará la implementación de Nova. La lección es que, si un método heredado no se comporta de la manera requerida, generalmente existe la opción de modificarlo con sobre escritura, para obtener un resultado deseado.

2.i) Imprime "Galaxia@19469ea2". Se puede ejecutar el método *toString* porque este es un método perteneciente a la clase *Object*, una clase de la cuál heredan todas las otras clases, y que no ha sufrido de ninguna modificación durante este proceso de ser heredado, por lo que retorna su valor predeterminado.

2.j) Porque no se está creando un objeto abstracto, obN es una referencia que está apuntando a un objeto tipo Nova.

2.k) La instrucción C es válida por polimorfismo, nova es simultáneamente, de tipo Nova y tipo *ObjetoAstronomicoExtraSolar*. Aplica casting.

La instrucción B, sin embargo, está mal; no se puede crear un objeto de una clase abstracta.

2.l) La instrucción B es válida por polimorfismo, nova es simultáneamente, de tipo Nova y tipo *ObjetoAstronomicoExtraSolar*. Aplica casting.

La instrucción C no lo es porque el método *explotar()* pertenece a la clase *Estrella*, no a *ObjetoAstronomicoExtraSolar*, oa es de tipo *ObjetoAstronomicoExtraSolar*.

Omitiendo C, D es valida e imprime "Boom!", por casting.

2.m) Imprime true porque obN hereda de la clase Object, por lo que es una instancia de esta. Todos los objetos heredan de Object, por lo que todos son una instancia de esta clase, e imprimirán true.

Las otras líneas imprimen:

```
false
true
```

obN de por si es un nulo, no hereda de Object y retorna false, pero al concatenarlo con un String vacío, como String si hereda de Object, el resultado retorna true.

2.n) No se provoca error, las clases abstractas pueden tener constructores, no para su uso propio, pero si para el uso de sus subclases.

2.o) La nueva clase presenta errores al no poseer los métodos abstractos de sus superclases, *Estrella* y *ObjetoAstronomicoExtraSolar* : *explotar* y *descripción*, respectivamente.

```
src > J EnanaBlanca.java > ...
1  class EnanaBlanca extends Estrella{
2      vo
3      }
4      }
5      }
6      EnanaBlanca
7      View Problem (Alt+F8) Quick Fix... (Ctrl+.)
8
9
```

La situación se puede corregir implementando estos métodos de manera parecida a la que se aprecia en la imagen:

```
src > J EnanaBlanca.java > ...
1  class EnanaBlanca extends Estrella{
2      void agotarCombustible(){
3          System.out.println(x:"Enana blanca muere");
4      }
5
6      void explotar() {
7          System.out.println(x:"¡Pam!");
8      }
9
10     void descripcion() {
11         System.out.println(x:"Soy una Enana blanca");
12     }
13 }
14
```