20231008_02

Herencia con interfaces

What will be the output of compiling and running the following program:

```
class TestClass implements I1, I2{
   public void m1() { System.out.println("Hello"); }
   public static void main(String[] args){
      TestClass tc = new TestClass();
      ( (I1) tc).m1();
   }
}
interface I1{
   int VALUE = 1;
   void m1();
}
interface I2{
   int VALUE = 2;
   void m1();
}
```

Please select 1 option

- It will print Hello.
- There is no way to access any VALUE in TestClass.
- The code will work fine only if VALUE is removed from one of the interfaces.
- It will not compile.
- None of the above.

Puntos claves:

- Características de una Interface:
  - No puede ser instanciada.
  - Por naturaleza los métodos de una interfaz son abstractos y públicos.
  - Todos los atributos definidos son públicos, estáticos y finales.
- Una clase puede implementar múltiples interfaces.
- Al ejecutar un método se considera el tipo de objeto, no su variable de referencia.

Análisis de opciones

Opción 1: It will print hello
Puntos a favor:

- Las interfaces se encuentran bien implementadas.
- Las interfaces se encuentran bien definidas.

- Se instancia un objeto de tipo TestClass en una variable de tipo TestClass.
- Se hace un cast correctamente a un tipo I1 adecuadamente.
- El método m1() se encuentra correctamente sobreescrito en TestClass.
- Se imprime hello.

Opción 2: There is no way to access any VALUE in TestClass

Puntos a favor:

- Existe una ambigüedad en la definición del atributo VALUE debido al nombre.

Puntos en contra:

- Es posible acceder al atributo VALUE si se realiza el cast correspondiente a una variable de tipo I1 ó I2.

Opción 3: The code will work fine only if VALUE is removed from one of the interfaces.

Puntos a favor:

- Existe una ambigüedad en la definición del atributo VALUE debido al nombre.

Puntos en contra:

- Es posible acceder al atributo VALUE si se realiza el cast correspondiente a una variable de tipo I1 ó I2.
- Es posible tener dos interfaces implementadas con esta ambigüedad, únicamente hay que cuidar los llamados a los atributos.

Opción 4: It will not compile.

Puntos a favor:

- Existe una ambigüedad en la definición del atributo VALUE debido al nombre.
- Es posible tener dos interfaces implementadas con esta ambigüedad, únicamente hay que cuidar los llamados a los atributos.

Puntos en contra:

- Las interfaces se encuentran bien implementadas.
- Las interfaces se encuentran bien definidas.
- Se instancia un objeto de tipo TestClass en una variable de tipo TestClass.
- Se hace un cast correctamente a un tipo I1 adecuadamente.
- El método m1() se encuentra correctamente sobreescrito en TestClass.
- Se imprime hello.

Opción 5: None of the above

Respuesta correcta:

Opción 1: It will print hello

Notas

La línea en donde se realiza el cast no es necesaria, pues se está invocando un método y para la invocación de métodos es importante considerar el objeto más que la variable de referencia.

La forma más adecuada para invocar un atributo es utilizando el nombre de la interfaz (I1.VALUE), pues los atributos son static final, por lo que pertenece a la interfaz.

Alternativas

- Si después de la instancia de *tc* imprimimos el valor de *VALUE* sin hacer cast, System.out.println(tc.VALUE()), el compilador manda error por considerar ambigua la solicitud, pues no se puede determinar a cuál de las interfaces se refiere.
- Para el caso anterior, puede determinarse a qué interfaz nos referimos realizando un cast; System.out.println(((I1)tc).VALUE())}

```java
package com.curso.domingo8.v2;

class TestClass extends C1 implements I2 {

    public static void main(String[] args) {
        TestClass tc = new TestClass();
        tc.m1();
    }

}

class C1 {
    int VALUE;
    public void m1() {
        System.out.println("C1");
    }
}

interface I2 {
    int VALUE = 2;
    void m1();
}
```

- }
- En este caso no es necesario definir m1() en la clase TestClass pues la clase C1 ya ha implementado ese método y es el que usa.

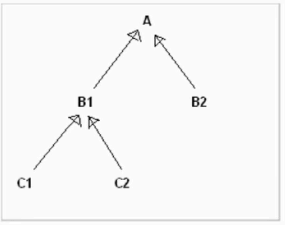
Herencia y sobreescritura

Consider the following class hierarchy shown in the image. (B1 and B2 are subclasses of A and C1, C2 are subclasses of B1)

Assume that method `public void m1(){ ... }` is defined in all of these classes EXCEPT B1 and C1.

Assume that "`objectOfXX`" means a variable that points to an object of class XX. So, `objectOfC1` means a reference variable that is pointing to an object of class C1.

Which of the following statements are correct?



Please select 1 option

- `objectOfC1.m1();` will cause a compilation error.
- `objectOfC2.m1();` will cause A's `m1()` to be called.
- `objectOfC1.m1();` will cause A's `m1()` to be called.
- `objectOfB1.m1();` will cause an exception at runtime.
- `objectOfB2.m1();` will cause an exception at runtime.

Puntos claves:

- Los métodos son heredados a las subclases.
- Al declarar un método en una subclase con la misma firma de un método e la superclase se realiza sobreescritura.
- Si no hay sobreescritura se utilizan los métodos heredados.

Análisis de opciones

Opción 1: objectOfC1.m1(); will cause a compilation error.
Puntos en contra:

- Al invocar m1 desde C1 se invoca al método definido en A.
- Toma el método de su abuelo.

Opción 2: objectOfC2.m1(); will cause A's m1() to be called.
Puntos en contra:

- Al invocar m1 desde C2 se invoca al método definido en C2.
- Toma el método que sobreescribe.

Opción 3: objectOfC1.m1(); will cause A's m1() to be called.
Puntos a favor:

- Al invocar m1 desde C1 se invoca al método definido en A.
- Toma el método de su abuelo.

Opción 4: objectOfB1.m1(); will cause an exception at runtime.
Puntos en contra:

- Al invocar m1 desde B1 se invoca al método definido en A.
- Toma el método de su abuelo, no hay exception.

Opción 5: objectOfB2.m1(); will cause an exception at runtime.
Puntos en contra:

- Al invocar m1 desde B2 se invoca al método definido en B2.
- Toma el método que sobreescribe, no hay exception.

Respuesta correcta:

Opción 3: objectOfC1.m1(); will cause A's m1() to be called.

Herencia de métodos y atributos

```
What will the following code print when compiled and run?
class Baap{
    public int h = 4;
    public int getH(){
        System.out.println("Baap "+h); return h;
    }
}


public class Beta extends Baap{
    public int h = 44;
    public int getH(){
        System.out.println("Beta "+h); return h;
    }
    public static void main(String[] args) {
        Baap b = new Beta();
        System.out.println(b.h+" "+b.getH());
        Beta bb = (Beta) b;
        System.out.println(bb.h+" "+bb.getH());
    }
}
```

Puntos claves:

- Cuando se invoca un atributo hay que poner atención a la referencia.
- Cuando se invoca un método hay que poner atención al objeto.

Análisis de código

- La clase Beta hereda de Baap
- Se instancia un Beta en una variable tipo Baap.
- Se imprime su parámetro h y el método getH(), el valor de h que se muestra es el del padre, h=4; el método getH() que se invoca es el correspondiente al objeto Beta.
- Dentro de getH() en Beta se imprime la palabra Beta seguido de la h correspondiente a Beta, entonces se obtiene: Beta 44, y a parte hay un return con valor de 44
- Es importante mencionar que primero se realiza la impresión dentro del método getH(), y una vez que se tiene el valor de retorno se imprime la linea del programa principal, de manera que hasta este punto del código nuestra salida es:
  Beta 44
  4 44
- En seguida se realiza el cast del objeto anterior hacia un tipo Beta.
- Se busca imprimir el atributo h del nuevo objeto y su método getH().
- El atributo h toma el valor de 44 y getH() imprime Beta 44 y retorna 44.
- La impresión final queda:

Beta 44
4 44
Beta 44
44 44

Respuesta correcta:

Beta 44
4 44
Beta 44
44 44