Nombre: ..... Fecha: / 11 / 2009

Grupo: 1 ☐ 2 ☐ 3 ☐ 4 ☐

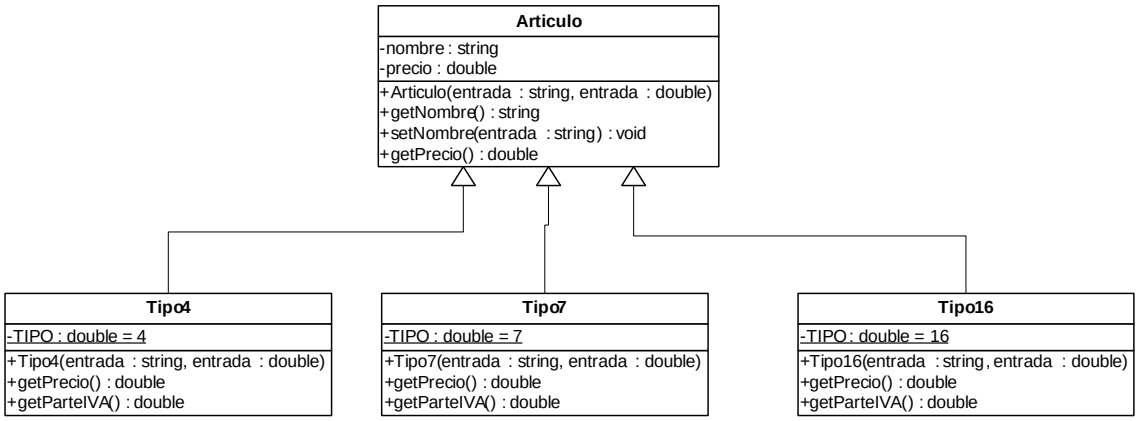
PRÁCTICA 8
HERENCIA ENTRE CLASES Y POLIMORFISMO DE MÉTODOS (II)

En esta práctica recuperaremos uno de los ejemplos que introdujimos en la Hoja 2 de ejercicios para seguir trabajando con la definición de *relaciones de herencia* tanto en Java como en C++. Este mismo ejemplo nos servirá para incidir de nuevo en la noción de *polimorfismo de métodos* también en ambos lenguajes.

En la segunda parte de la práctica aprenderemos alguno de lo métodos relevantes dentro de la librería de Java, lo adaptaremos a las clases que hemos programado en la primera parte, observaremos de nuevo su comportamiento polimorfo y trataremos de adaptar el mismo ejemplo a C++.

Parte 1. Relaciones de herencia y polimorfismo de métodos.

Recuperamos el siguiente diagrama de clases en UML que introdujimos en la Hoja de Ejercicios 2.



Los métodos especificados tienen el comportamiento esperado. El método "getPrecio(): double" en la clase **Articulo** devuelve el atributo "precio: double". En las clases derivadas (**Tipo4**, **Tipo7**, **Tipo16**) devuelve el valor de "precio: double" más el IVA correspondiente. El método "getParteIVA(): double" devuelve el resultado de multiplicar el valor de "TIPO: double" por "precio:double" y dividirlo por 100.

1. Programa el anterior diagrama de clases en C++ y Java. Observa que el método "getPrecio(): double" es *polimorfo*. Tenlo en cuenta a la hora de declararlo en C++ en el fichero "Articulo.h".

2. Declara un programa cliente del anterior sistema de clases en forma de una función "main" (créala en un fichero de nombre "principal_prac8") tanto en Java como en C++ que realice las siguientes operaciones:

a) Declara un objeto "art1" de la clase **Articulo**.
b) Crea el anterior artículo por medio del constructor "Tipo4(string, double)" con nombre "La historia interminable", y precio "9" euros.
c) Volvemos a incidir en la diferencia entre *declaración* y *construcción*. Trata de acceder al método "getParteIVA(): double" sobre "art1". Observa el resultado (deja la anterior sentencia comentada).
d) Declara un objeto "art2" de la clase **Tipo7**.
e) Crea el anterior artículo por medio del constructor "Tipo7(string, double)" con nombre "Gafas", y precio "160" euros.
f) Comprueba cuál es la parte correspondiente al IVA de este artículo (método "getParteIVA(): double") y observa la diferencia con el comportamiento sobre "art1".
g) Declara un objeto "art3" de la clase **Articulo** y constrúyelo por medio del constructor de la clase **Tipo16** con valores "Bicicleta" y precio "550" euros.

h) ¿Qué resultado obtendrás si intentas invocar al método "getParteIVA(): double" sobre "art3"?

3. Veamos si los objetos que hemos declarado y construido son capaces de realizar *enlazado dinámico* de métodos, y, por tanto, acceder a sus métodos de *modo polimorfo*:

a) Invoca al método "getPrecio(): double" sobre "art1" y observa el resultado ¿Ha habido enlazado dinámico (y por tanto polimorfismo) en la llamada a "getPrecio(): double" en Java y en C++?

b) Invoca al método "getPrecio(): double" sobre "art2" y observa el resultado ¿Era necesario el enlazado dinámico en la llamada a "getPrecio(): double"?

c) Invoca al método "getPrecio(): double" sobre "art3" y observa el resultado ¿Ha habido enlazado dinámico (y por tanto polimorfismo) en la llamada a "getPrecio(): double" en Java y en C++?

4. Vamos a detenernos un poco más en las propiedades del polimorfismo. Programa el método "getPrecio(): double" en todas las clases derivadas por medio de la siguiente definición (en Java, hazlo de modo similar en C++):

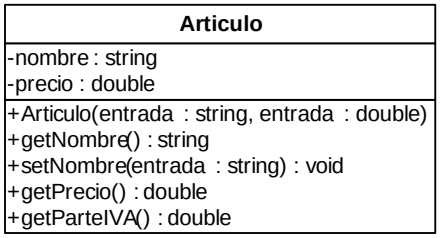
```
public double getPrecio (){
                return (super.getPrecio() + this.getParteIVA());
}
```

Repite las invocaciones realizadas en el ejercicio 3 al método "getPrecio(): double".

Recuerda el resultado que hemos obtenido en los anteriores ejercicios, especialmente en los puntos 2 c) y h). Los objetos "art1" y "art3" no han podido acceder **directamente** al método "getParteIVA(): double", al no ser declarados de las clases derivadas **Tipo4** y **Tipo7**. Sin embargo, sí han sido capaces de hacerlo desde la definición del método "getPrecio(): double" de sus respectivas clases.

5. Vamos a aumentar la presencia de métodos polimorfos en nuestro diagrama de clases. Como hemos comentado en los puntos 2 c) y h), no es posible acceder al método "getParteIVA(): double" desde los objetos *declarados* de la clase **Articulo**.

Vamos a intentar solventar esta situación por medio de una solución un tanto anómala. Vamos a "reprogramar" la clase **Articulo** con respecto al siguiente diagrama UML:



Como en la clase **Articulo** todavía no podemos **definir** de modo correcto el comportamiento de "getParteIVA(): double", optaremos por una solución poco satisfactoria, que es declararlo de tal modo que devuelva siempre "0.0".

En el Tema 4 veremos una forma más correcta de solventar la situación anterior, por medio de **métodos abstractos**.

6. Observa que ahora el método "getParteIVA(): double" está definido en la clase base **Articulo** y *redefinido* en sus clases derivadas **Tipo4**, **Tipo7** y **Tipo16**. Realiza los cambios necesarios en C++ para que dicho método se comporte de modo polimorfo.

Comprueba que ahora en C++ puedes invocar al método "getParteIVA(): double" desde los objetos "art1", "art2" y "art3". Comprueba que el resultado de las anteriores invocaciones es el mismo que obtienes en Java.

7. Comprueba que además "getPrecio(): double" se sigue comportando de modo polimorfo sobre "art1", "art2" y "art3", tanto en Java como en C++. Recuerda que en ambos lenguajes debes obtener los mismos resultados.

Parte 2. Redefinición de métodos de la librería de Java. Soluciones alternativas en C++.

Al hablar de la herencia en el Tema 2 mencionamos que en Java, todas las clases definidas por los usuarios heredan de una clase propia del lenguaje llamada **Object**. Busca la especificación de dicha clase en la API de Java 1.6 (http://java.sun.com/javase/6/docs/api/java/lang/Object.html).

8. Para comprobar la anterior afirmación, realiza en Java las siguientes invocaciones sobre los objetos "art1", "art2" y "art3":

a) Comprueba, con el método "equals(Object): boolean" y muestra por pantalla si "art1" es igual que "art2".
b) Comprueba con el método "getClass (): Class" y muestra por pantalla la clase a la que pertenecen los objetos "art1", "art2" y "art3".
c) Comprueba y muestra por pantalla el resultado de invocar al método "toString(): String" sobre los objetos "art1", "art2" y "art3".

9. Si lees detalladamente la especificación del método "toString(): String" que se hace en la API de Java (http://java.sun.com/javase/6/docs/api/java/lang/Object.html#toString()) observarás que se recomienda *redefinir* ("override" en inglés) el mismo en todas y cada una de nuestras clases.

Redefine el método "toString(): String" en la clase **Articulo** en Java de tal modo que muestre la siguiente cadena de texto:

"El objeto actual pertenece a la clase " + this.getClass() + "\n" +
"El objeto tiene por nombre "+ this.getNombre () + "\n" +
"El objeto tiene por precio " + this.getPrecio() + " euros " + "\n"

Redefine el método "toString(): String" en las clases **Tipo4**, **Tipo7** y **Tipo16** para que dicho método llame al método "toString(): String" de la clase base y, además, muestre por pantalla:

"El objeto tiene un tipo de IVA del " + TIPO + "\n" +
"La parte de su precio correspondiente al IVA es " + this.getParteIVA() + "\n"

10. Declara y crea un objeto de la clase **Articulo** de nombre "art4" con nombre "Armario" de precio "350" euros.

11. Repasamos ahora el uso de métodos polimorfos desde funciones auxiliares. Define una función auxiliar:

```
public static void mostrarObjeto(Object obj){
        System.out.println(obj.toString());
}
```

Comprueba el resultado de llamar a la función "mostrarObjeto(Object): void" con los objetos "art1", "art2", "art3" y "art4" ¿Has obtenido comportamiento polimorfo? Observa que, como en el Ejercicio 4, en las redefiniciones del método "toString(): String", definido originalmente en la clase **Object**, hemos podido acceder a los métodos "getNombre(): string", "getPrecio(): double", "getParteIVA(): double", ... , que no existen en la clase **Object**.

12. Vamos a aprender un nuevo uso del modificador **"final"**, en el caso de que éste vaya referido a un método (recuerda que en Java ya lo hemos usado para definir atributos que no pueden ser modificados una vez inicializados). Declara con el modificador "final" el método "toString(): String" en la clase **Articulo**. Compila el proyecto. Observa el mensaje de error obtenido ¿Qué hace el modificador **"final"**? Deja el modificador entre comentarios.

13. En C++ no disponemos de una clase **Object**, ni, por tanto, de un método "toString(): String" como en Java. Sin embargo podemos idear una solución similar a la aplicada en Java.

Define un método "toString(): string" (recuerda que el tipo "string" en C++ lo puedes codificar por "char []" ó "char *") en C++ en la clase **Articulo** que se comporte como hemos especificado en el ejercicio 9 ¿Cómo puedes suplir la llamada al método "getClass (): Class" propia de Java?

Redefine el método "toString(): string" en las clases **Tipo4**, **Tipo7** y **Tipo16** de forma que se comporten como hemos especificado en el ejercicio 9 ¿Cómo puedes suplir de nuevo las llamadas al método "getClass (): Class"?

14. Repite el ejercicio 10 en C++

15. Repite el ejercicio 11 en C++. Define una función auxiliar:

```
void mostrarArticulo(Articulo & art){
        cout << art.toString();
}
```

Debes obtener el siguiente resultado de la ejecución:

```
El objeto actual pertenece a la clase Tipo4
El objeto tiene por nombre La historia interminable
El objeto tiene por precio 9.36 euros
El objeto tiene un tipo del IVA 4.0
La parte de su precio correspondiente al IVA es 0.36 euros

El objeto actual pertenece a la clase Tipo7
El objeto tiene por nombre Gafas
El objeto tiene por precio 171.2 euros
El objeto tiene un tipo del IVA 7.0
La parte de su precio correspondiente al IVA es 11.2 euros

El objeto actual pertenece a la clase Tipo16
El objeto tiene por nombre Bicicleta
El objeto tiene por precio 638.0 euros
El objeto tiene un tipo del IVA 16.0
La parte de su precio correspondiente al IVA es 88.0 euros

El objeto actual pertenece a la clase Articulo
El objeto tiene por nombre Armario
El objeto tiene por precio 350.0 euros
```

ENTREGA los archivos *Articulo.cpp*, *Articulo.h*, *Tipo4.cpp*, *Tipo4.h*, *Tipo7.cpp*, *Tipo7.h*, *Tipo16.cpp*, *Tipo16.h*, *principal_prac8.cpp*, *Articulo.java*, *Tipo4.java*, *Tipo7.java*, *Tipo16.java*, *principal_prac8.java*. Los archivos deben contener **todos los cambios** realizados durante los ejercicios de la práctica. La entrega es a través de Belenus en el repositorio de Julio Rubio. También deben contener unas cabeceras que permitan identificarte:

```
/* Nombre: ___
     Grupo: _____
     Nombre del fichero: ____
     Descripción: _______*/
```

Guarda una copia de los ficheros para las prácticas sucesivas