Nombre: ..... Fecha: / 11 / 2008

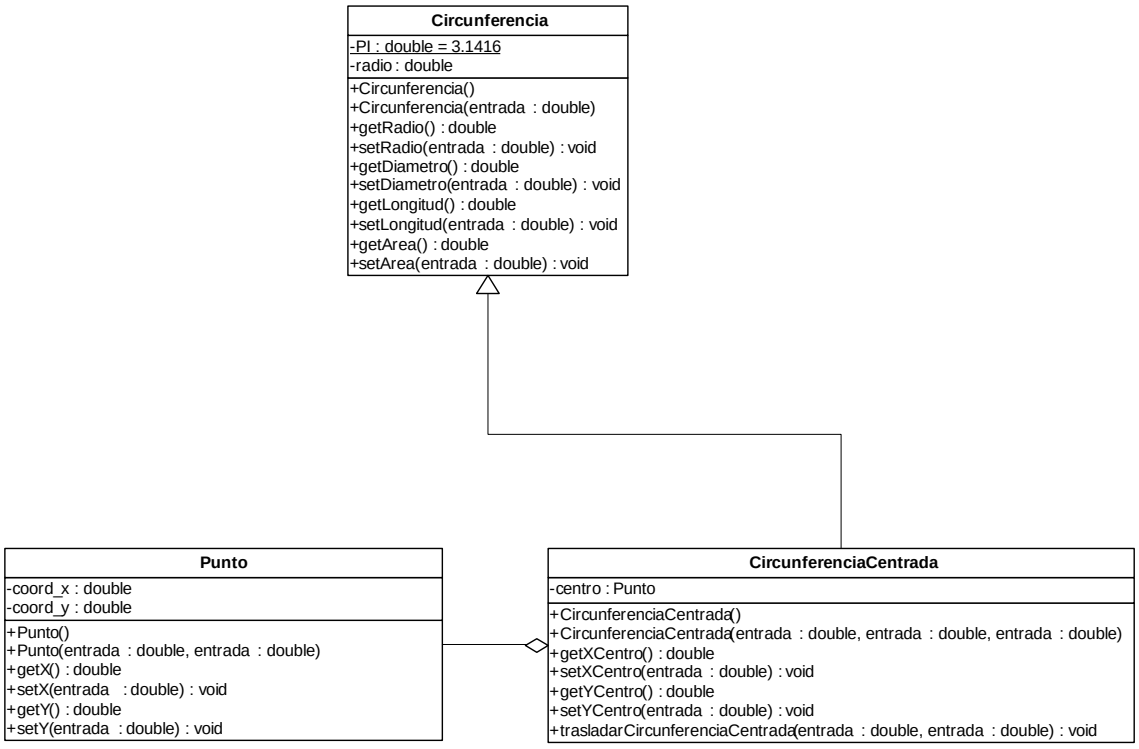
Grupo: 1 ☐ 2 ☐ 3 ☐ 4 ☐

PRÁCTICA 5
RELACIONES DE HERENCIA ENTRE CLASES EN JAVA Y C++

En esta práctica se pretende que el alumno comience a trabajar en la definición y uso de clases definidas a partir de otras ya existentes por medio de relaciones de herencia y observe las ventajas de la ***reutilización*** de componentes ya programadas. Asimismo, introduciremos el modificador de acceso "protected" y alguna de sus posibilidades de uso. También ilustraremos diversos casos de redefinición de métodos que nos permitan presentar las ideas de "hijos buenos" e "hijos malos" que, en la Práctica 6, nos ayudarán a introducir la noción de polimorfismo. Para ello nos basaremos en algunos de los ejemplos de la asignatura que hemos presentado en las prácticas anteriores.

Crea en Dev-C++ un proyecto llamado "Practica5_parte1"; crea en JCreator un "Workspace" llamado Practica5, y dentro del mismo un proyecto de nombre "Practica5_parte1".

1. Traduce el siguiente diagrama UML a C++ y Java, definiendo el comportamiento de cada uno de los métodos, en ficheros "CircunferenciaCentrada.h", "CircunferenciaCentrada.cpp", y "CircunferenciaCentrada.java". Puedes recuperar las clases "Punto" y "Circunferencia" de prácticas anteriores.



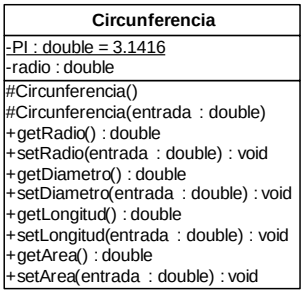
Según el diagrama anterior, un objeto de la clase "CircunferenciaCentrada" "tiene - un" objeto de la clase "Punto" (relación de asociación) y "es - un" objeto de la clase "Circunferencia" (relación de herencia).

2. Recupera el guión de la Práctica 4:

- Observa en el diagrama de clases UML las diferencias entre la representación de la clase "CircunferenciaCentrada" en la práctica 4 y la que acabamos de presentar.
- Observa también la diferencia de uso dentro de la clase "CircunferenciaCentrada" de los atributos o métodos provenientes de la relación de asociación, "this.centro.getX ()", por ejemplo, y el uso de los métodos provenientes de la relación de herencia "this.getRadio()".

3. Recupera el programa cliente de la clase "CircunferenciaCentrada" que desarrollamos en la Práctica 4 (que guardamos como "Principal_prac4_1"), renómbralo como "Principal_prac5_1" y comprueba que obtienes el mismo resultado de ejecución que el que se obtenía.

4. Veamos ahora la utilidad del modificador de acceso "protected". Sobre el proyecto anterior, haz las modificaciones pertinentes en la clase "Circunferencia" (es decir, sustituir el modificador de acceso "public" por "protected" en ambos constructores de la clase) para que la clase "Circunferencia" se corresponda con el siguiente diagrama UML.



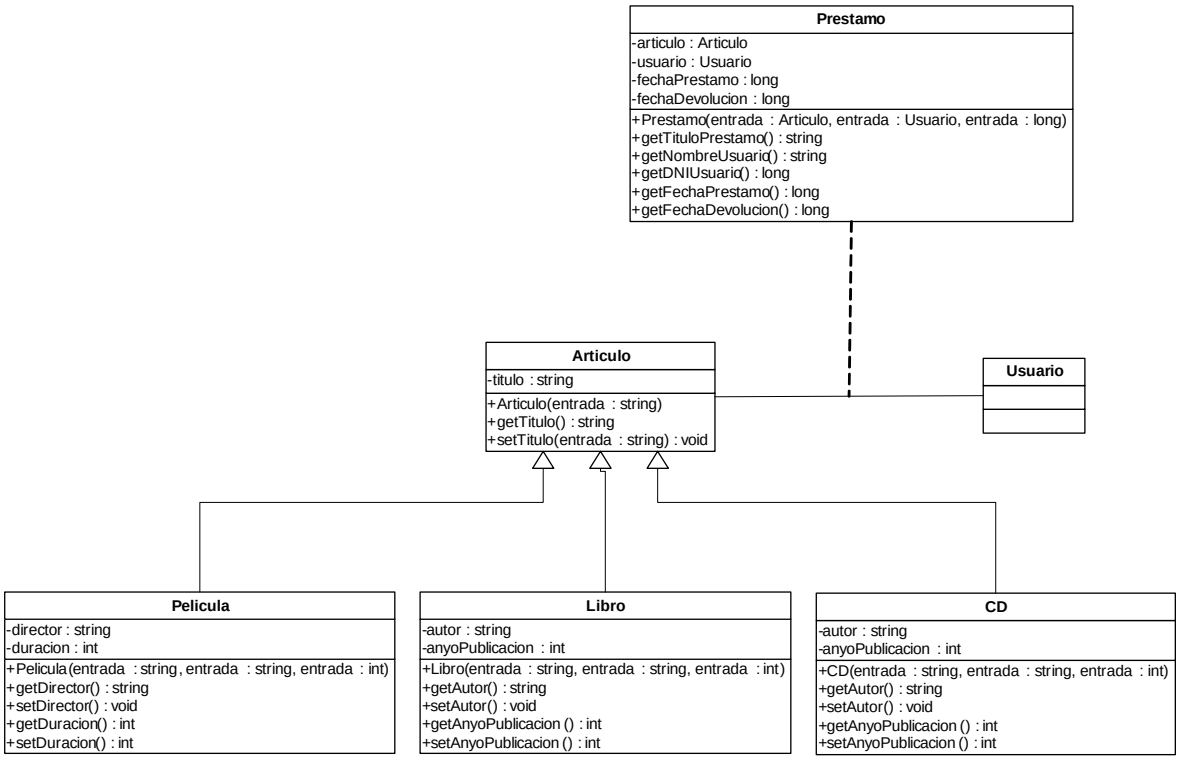
Compila el código del proyecto (debería compilar como anteriormente).

En el programa Principal_prac5_1, trata de declarar y construir objetos de la clase "Circunferencia", uno haciendo uso del constructor por defecto "Circunferencia()" y otro haciendo uso del constructor "Circunferencia(entrada: double)" ¿Qué sucede? (Escribe las líneas que producen el error entre comentarios en el código fuente).

¿Dónde son visibles ahora los constructores de la clase "Circunferencia"? ¿Para qué podremos usar la misma? ¿Podremos usarla en relaciones de asociación, agregación, ...?

Crea un nuevo proyecto en Java dentro del Workspace "Practica5" llamado "Practica5_parte2" y un proyecto en Dev-C++ del mismo nombre.

5. Recuperamos ahora el ejemplo sobre las clases "Usuario" y "Pelicula" tal y como lo desarrollamos en la Práctica 4. La oferta del vídeo-club se ha ampliado, incluyendo también artículos tales como libros y CD's de música. Ante la nueva situación, realizamos el siguiente diagrama de clases UML que nos permite salvar parte de nuestro desarrollo original:



Programa las clases correspondientes ("Articulo", "Libro" y "CD") que te permitan reproducir el mismo. Modifica asimismo la clase "Prestamo" para que se ajuste a la nueva situación (ya no posee como atributo un objeto de la clase "Pelicula", sino un objeto de la clase más general, "Articulo").

6. Realiza un programa principal (Principal_prac5_2), que ejerza de cliente del diagrama de clases anterior, en el que se simulen las siguientes situaciones:

a) Declara un objeto "usu1" de la clase "Usuario" y constrúyelo con nombre "Juan Mediano" y DNI 55444888.
b) Declara un objeto "art1" de la clase "Articulo" y constrúyelo, por medio del constructor de la clase "Pelicula", con título "El hombre que pudo reinar", director "John Huston", y duración 129 min.
c) Declara un objeto "art2" de la clase "Articulo" y constrúyelo, por medio del constructor de la clase "CD", con título "Kind of Blue", autor "Miles Davis" y año de publicación 1959.
d) Crea un objeto "prest1" de la clase "Prestamo", y constrúyelo con los parámetros usu1, art1 y fecha de préstamo "20081109".
e) Crea un objeto "prest2" de la clase "Prestamo", y constrúyelo con los parámetros usu1, art2 y fecha de préstamo "20081109".

7. Realiza la siguiente comprobación sobre el programa anterior; modifica el nombre del Usuario (por medio del método "setNombre(): void") y defínelo como "Antonio Redondo". Comprueba el valor de las invocaciones "prest1.getNombre()" y "prest2.getNombre()" ¿Qué valor esperabas obtener? ¿Qué valor has obtenido? Razona tus respuestas en relación a la copia o asignación de objetos que has hecho en el constructor de la clase "Prestamo".

Todos los ejemplos que hemos desarrollado hasta ahora se corresponden a casos de lo que se conoce en POO como "hijos buenos" (del inglés "good-child"), que quiere decir que todos los métodos heredados mantienen su comportamiento.

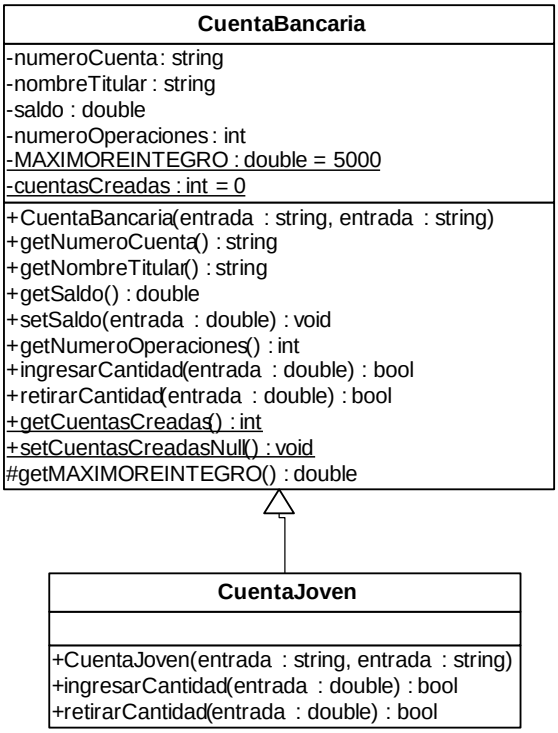
Puede haber casos en los que la clase "hija" deba reescribir alguno de los métodos de la clase "base". Hablaremos entonces de redefinición de métodos y de "hijos malos" (del inglés,"bad-child"). Veamos un ejemplo de lo mismo en el siguiente ejercicio.

Crea un proyecto de nombre "Practica5_parte3" dentro del "Workspace" "Practica5" en Java, y un nuevo proyecto "Practica5_parte3" en Dev-C++.

8. Recuperamos el ejemplo que proponíamos en la Hoja de Ejercicios 1 sobre una clase que representa un cuenta bancaria (aunque modificaremos ligeramente su especificación). El comportamiento de la clase "CuentaBancaria" es el esperado para cada uno de sus métodos; los métodos "retirarCantidad (entrada: double): bool" e "ingresarCantidad(entrada: double): bool" devuelven un booleano que dice si la operación se ha realizado con éxito o no. El método "retirarCantidad(entrada: double): bool" debe comprobar dos condiciones: que la cantidad retirada no sea mayor que "saldo" ni mayor que "MAXIMOREINTEGRO". Debes tener en cuenta que el atributo "numeroOperaciones: int" sólo contabiliza operaciones válidas (aquéllas cuyo resultado sea "true").

Queremos ahora definir una nueva clase llamada "CuentaJoven" que presenta las siguientes diferencias con respecto a su clase base, "CuentaBancaria":

a) La cantidad máxima que se puede retirar de la cuenta es un 25% de lo estipulado para objetos de la clase "CuentaBancaria" (dada por la constante MAXIMOREINTEGRO).
b) La cantidad máxima que se puede ingresar en la misma es también un 25 % de MAXIMOREINTEGRO.



¿Se podría haber programado la clase "CuentaJoven" sin hacer uso del método "getMAXIMOREINTEGRO(): double"?

9. Escribe ahora un programa principal (Principal_prac5_3) que haga de cliente de las clases anteriores y que simule las siguientes situaciones:

a) Declara un objeto (cuenta1) de la clase "CuentaBancaria" y constrúyelo con nombre "Eva Moreno" y cuyo número de cuenta bancaria sea "55552244441234567890".
b) Declara un objeto (cuenta2) de la clase "CuentaJoven" a nombre de "Miguel Moreno" cuyo número de cuenta bancaria sea "55552244440987654321".
c) Realiza un ingreso en ambas cuentas (método "ingresarCantidad (entrada: double): bool") de 1000 euros ¿Qué booleano devuelve la operación?
d) Realiza un segundo ingreso en ambas cuentas (método "ingresarCantidad (entrada: double): bool"), ahora de 2000 euros ¿Qué booleano devuelve la operación? ¿Qué valor tiene el atributo "numeroOperaciones: int" en ambos objetos?
e) Realiza un tercer ingreso en la cuenta "cuenta2" de 900 euros.
f) Realiza ahora un reintegro en ambas cuentas de 6000 euros. Comprueba que el resultado obtenido es el esperado ("false", las cuentas siguen teniendo el mismo saldo, y el atributo "numeroOperaciones: int" no ha sufrido alteración alguna).
g) Realiza un nuevo reintegro de 1500 euros en cada cuenta ¿Cuál es el resultado de la operación al operar con "cuenta1"? ¿Cuál es el resultado de la operación al operar con "cuenta2"?

ENTREGAR los archivos *CircunferenciaCentrada.cpp, CircunferenciaCentrada.h, Punto.cpp, Punto.h, Circunferencia.cpp, Circunferencia.h, Principal_prac5_1.cpp, CircunferenciaCentrada.java, Punto.java, Circunferencia.java, Principal_prac5_1.java, Articulo.cpp, Articulo.h, Pelicula.cpp, Pelicula.h, Libro.cpp, Libro.h, CD.cpp, CD.h, Usuario.cpp, Usuario.h, Prestamo.cpp, Prestamo.h, Principal_prac5_2.cpp, Articulo.java, Pelicula.java, Libro.java, CD.java, Usuario.java, Prestamo.java, Principal_prac5_2.java, CuentaBancaria.cpp, CuentaBancaria.h, CuentaJoven.cpp, CuentaJoven.h, Principal_prac5_3.cpp, CuentaBancaria.java, CuentaJoven.java, Principal_prac5_3.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