Nombre: ..... Fecha: / 10 / 2008

Grupo: 1 ☐ 2 ☐ 3 ☐ 4 ☐

PRÁCTICA 4
RELACIONES ENTRE CLASES: DEFINICIÓN DE NUEVAS CLASES DE CLASES EXISTENTES

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, y observe las ventajas de la ***reutilización*** de componentes ya programadas. Los tipos de relaciones entre clases que se contemplarán en la práctica son asociación, agregación y composición.

Crea en Dev-C++ un proyecto llamado "Practica4_parte1"; crea en JCreator un "Workspace" llamado Practica4, y dentro del mismo un proyecto de nombre "Practica4_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".

CircunferenciaCentrada
- PI: double = 3.1416 - radio: double - coord_x: double - coord_y: double
+ CircunferenciaCentrada() + CircunferenciaCentrada(double, double, double) + getRadio(): double + setRadio(double): void + getDiametro(): double + **setDiametro**(double):void + getLongitud(): double + setLongitud(double): void + getArea(): double + **setArea**(double):void + getXCentro(): double + setXCentro(double): void + getYCentro(): double + **setYCentro**(double):void + trasladarCircunferenciaCentrada(double, double): void

(Nota: el comportamiento de los métodos es el esperado. En el caso del método "trasladarCircunferenciaCentrada(double, double): void" consideraremos que los dos parámetros double se deben añadir a los atributos coord_x y coord_y que definen el centro)

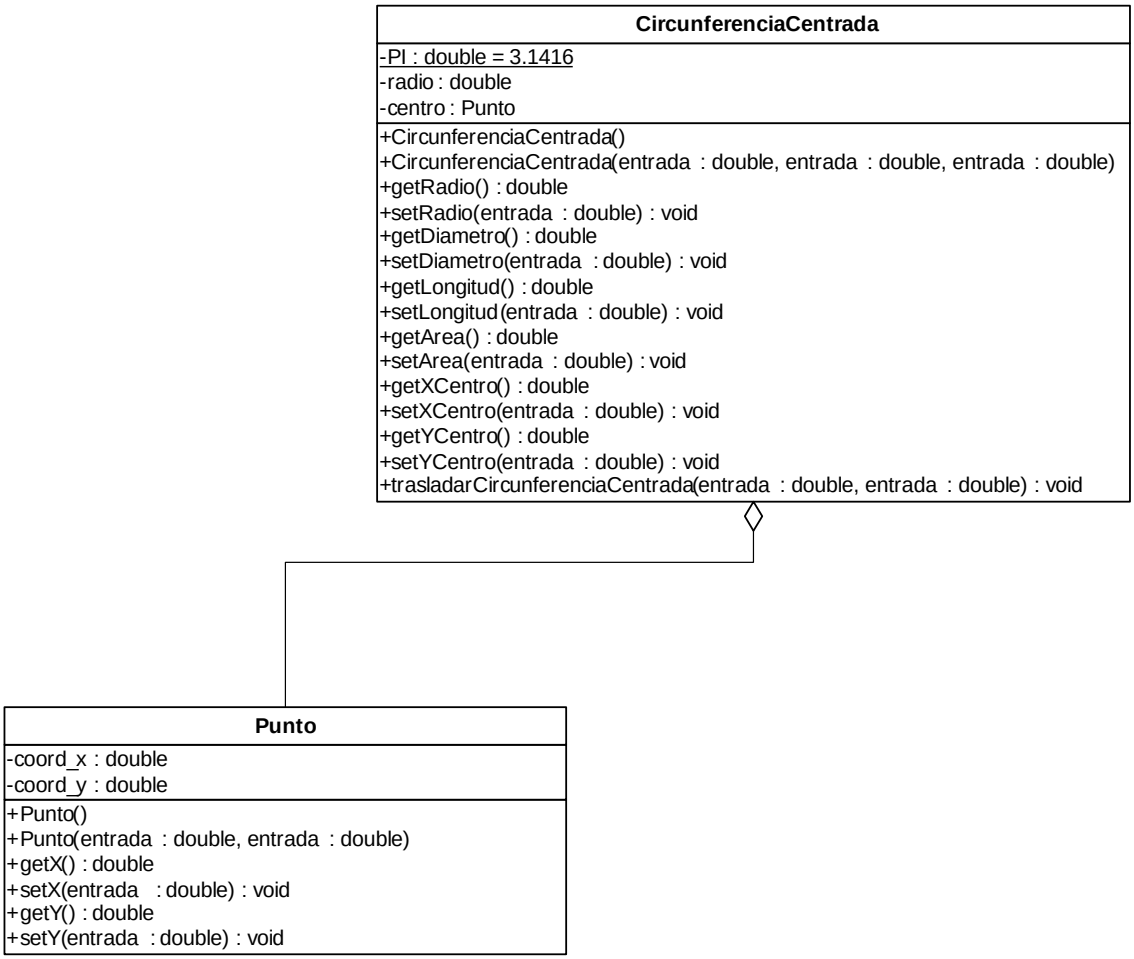
2. Crea (en C++ y Java) un programa cliente en un fichero "Principal_prac4_1" de nuestra clase "CircunferenciaCentrada" que genere las siguientes instancias de la misma:

- Una circunferencia centrada (circ1) de radio 8.0 y centro x = 3, y = 6
- Una circunferencia centrada (circ2) de radio 6.0 y centro x = 5, y = 5
- Una circunferencia centrada (circ3) de radio 6.5 y centro x = 0, y = 0
- Una circunferencia centrada (circ4) de radio 8.0 y centro x = 3, y = 6

3. Aplica diversas traslaciones a las circunferencias creadas. En particular, aplica traslaciones a las circunferencias circ1 y circ2 que permitan disminuir sus coordenadas x e y a la mitad de su valor.

4. Vamos a proponer ahora una solución alternativa al problema anterior. Crea un nuevo proyecto llamado "Practica4_parte2" tanto en JCreator como en C++. Define ficheros "CircunferenciaCentrada.h", "CircunferenciaCentrada.cpp" y "CircunferenciaCentrada.java". Recupera la clase **Punto** que definiste en las Prácticas 1 y 2.

Implementa el siguiente diagrama UML:



En el anterior diagrama UML hemos definido la clase **CircunferenciaCentrada** por medio de una relación de *Agregación* a la clase **CircunfenciaCentrada** de la clase **Punto**.
(Nota: la definición de los parámetros como "entrada" sólo quiere decir que los mismos no serán utilizados para devolver ningún valor; la decisión de transferirlos por valor o referencia recae en el programador de la clase).

Observa el siguiente hecho y propón una respuesta al mismo:

¿Cómo has tenido que acceder a los atributos "coord_x" y "coord_y" desde la definición de la clase **CircunferenciaCentrada**? ¿Por qué?

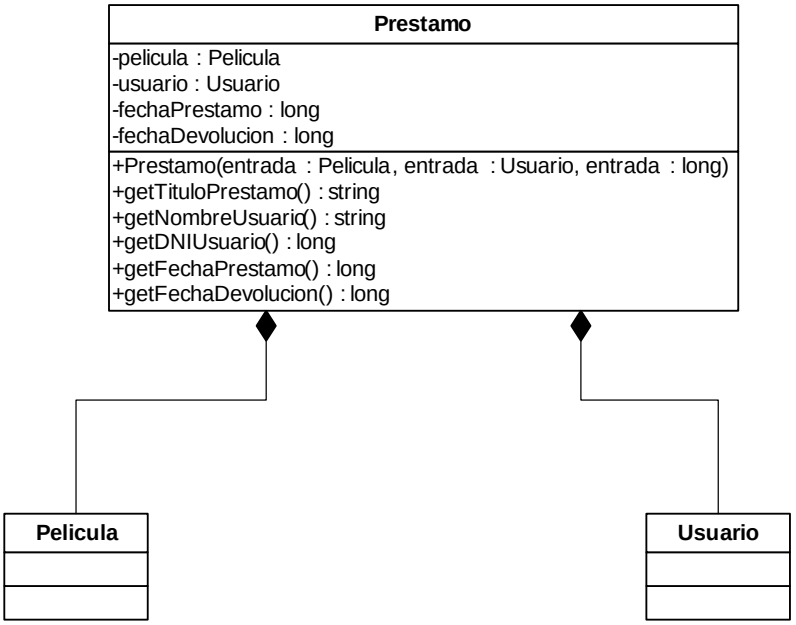
Respuesta:

5. Crea un fichero de nombre Principal_prac4_2 y repite los ejercicios 2 y 3 con la nueva implementación que has hecho de la clase **CircunferenciaCentrada**.

6. Vamos a repasar el concepto de "ocultación de la información" que proponíamos en el Tema 1 y en la Práctica 3. Recupera la implementación de la clase **Punto** que realizaste en la Práctica 3 por medio de coordenadas polares. Crea un nuevo proyecto en Java y en C++ ("Practica4_parte3") y reemplaza en el mismo la clase **Punto**, sustituyéndola por su implementación con coordenadas polares (recuerda que debes implementar el método "trasladarCircunferenciaCentrada (double, double): void" en la misma).

Verifica que el cliente de la misma clase (en este caso, la clase **CircunferenciaCentrada**) no ha percibido el cambio de implementación realizado. Para verificarlo puedes realizar diversas llamadas a los métodos de la misma que demuestren que el comportamiento de la clase **CircunferenciaCentrada** no ha sufrido modificaciones.

7. Vamos a ver un nuevo tipo de relación entre clases. Crea dos proyectos en JCreator y en Dev-C++ de nombre "Practica4_parte4". Recupera los ficheros de las Prácticas 1 y 2 donde implementaste las clases **Película** y **Usuario** en Java y C++ y define en ambos lenguajes el siguiente diagrama UML:



En el diagrama anterior se muestra un caso de definición de una clase (**Prestamo**) por *Composición* de otras clases ya preexistentes (**Pelicula** y **Usuario**). La "fechaPrestamo" es un número en el formato aaaammdd (20080417 representaría el día 17 de Abril de 2008). El atributo "fechaDevolucion" se calcula como el atributo "fechaPrestamo" más dos días (o más dos unidades, en enteros largos).

Como se puede observar, la clase **Prestamo** sólo permite métodos de acceso a los distintos atributos, ya que una vez realizado un préstamo no se supone que el mismo sea modificable (no dispone de métodos "set..()").

8. Crea un programa principal en ficheros de nombre "Principal_prac4_4.java" y "Principal_prac4_4.cpp" que haga lo siguiente:

a) Declara un objeto de la clase película e inicialízalo con los siguientes atributos: título "La vida de Brian", director "Terry Jones", duración 94 min

b) Declara un objeto de la clase Usuario e inicialízalo con nombre "Antonio Rojo" y DNI 55555555

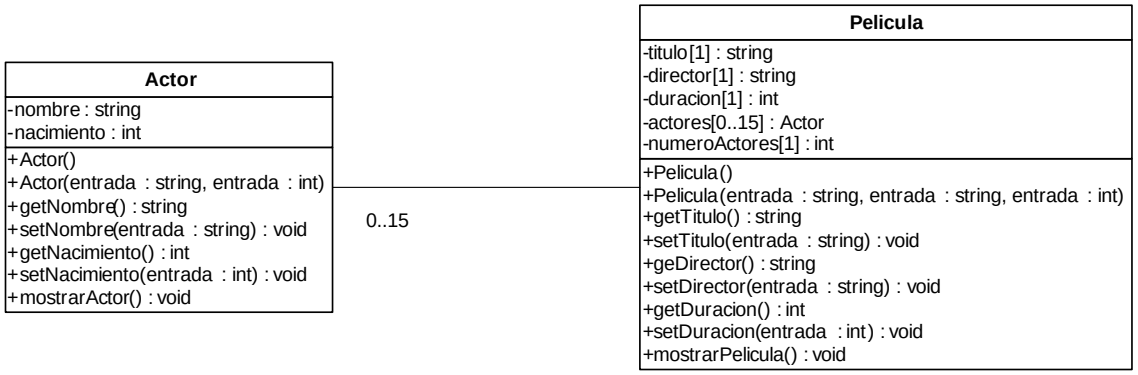
c) Declara un objeto de la clase Préstamo e inicialízalo con el objeto que has creado en a), el objeto creado en b) y con fecha la actual

d) Declara un objeto de la clase película e inicialízalo con los siguientes atributos: título "Los caballeros de la mesa cuadrada", director "Terry Gillian", duración 91 min

e) Declara un objeto de la clase Préstamo e inicialízalo con el objeto que has creado en d), el objeto creado en b) y con fecha la actual

9. Crea un nuevo proyecto "Practica4_parte5" en JCreator y en Dev-C++. En él vamos a presentar un tercer tipo de relación entre clases, conocido como *Asociación*. Recupera la

declaración y definición de la clase **Pelicula** que realizaste en las Prácticas 1 y 2 e introduce las modificaciones necesarias para definir el siguiente diagrama UML:



El anterior gráfico nos indica que la clase **Pelicula** debe incluir un nuevo atributo que contenga información sobre un grupo de actores (entre 0 y 15). Llama a este nuevo atributo "actores" y defínelo de tipo "array" tanto en Java como en C++. Define un segundo atributo adicional, numeroActores, que te permita saber cuántos actores tiene asociados una película. El nuevo método "mostrarPelicula (): void" debe mostrar por pantalla datos sobre el título, director, duración, y la lista de actores de una película.

Define en la clase **Pelicula** métodos que te permitan realizar las siguientes operaciones:

a) "isActor (Actor): bool": comprueba si en una película determinada el actor que se pasa como parámetro ha participado
b) "introduceActor (Actor): void": introduce un nuevo actor en una película (siempre que el número de actores en la misma no sea ya 15)

10. Define un programa principal en un fichero "Principal_prac4_5" que haga uso de la clase anterior. Declara e inicializa un objeto de la clase **Pelicula** de título "La guerra de las Galaxias", cuyo director sea "George Lucas" y su duración 121 minutos.

Define e inicializa objetos de la clase **Actor** con los valores "Mark Hamill", 1951, "Harrison Ford", 1942, "Carrie Fisher", 1956 y "Alec Guinness", 1914.

Introduce los objetos anteriores en el objeto de la clase **Pelicula** que habías creado.

Muestra por pantalla el objeto de la clase **Pelicula** que acabas de definir.

Comprueba si el objeto de la clase **Actor** representando a Alec Guinness tomó parte en la película.

ENTREGAR los archivos *CircunferenciaCentrada.cpp*, *CircunferenciaCentrada.h*, *Principal_prac4_1.cpp*, *CircunferenciaCentrada.java*, *Principal_prac4_1.java*, *CircunferenciaCentrada.cpp*, *CircunferenciaCentrada.h*, *Punto.cpp*, *Punto.h*, *Principal_prac4_2.cpp*, *CircunferenciaCentrada.cpp*, *CircunferenciaCentrada.h*, *Punto.cpp*, *Punto.h*, *Principal_prac4_3.cpp*, *CircunferenciaCentrada.java*, *Punto.java*, *Principal_prac4_3.java*, *Pelicula.cpp*, *Pelicula.h*, *Usuario.cpp*, *Usuario.h*, *Prestamo.cpp*, *Prestamo.h*, *Principal_prac4_4.cpp*, *Pelicula.java*, *Usuario.java*, *Prestamo.java*, *Principal_prac4_4.java*, *Pelicula.cpp*, *Pelicula.h*, *Actor.cpp*, *Actor.h*, *Principal_prac4_5.cpp*, *Pelicula.java*, *Actor.java*, *Principal_prac4_5.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