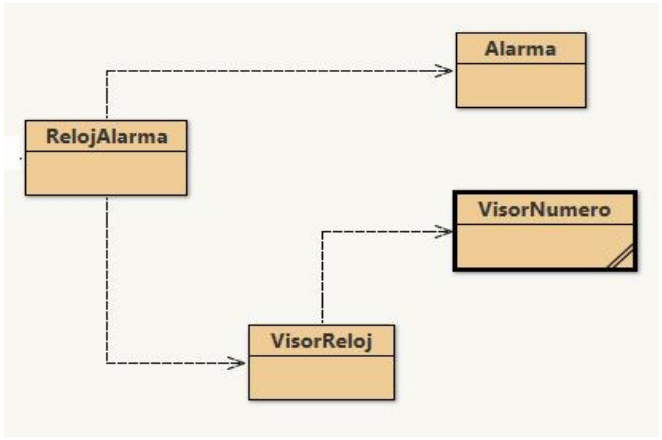
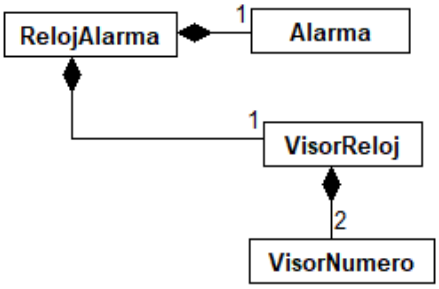
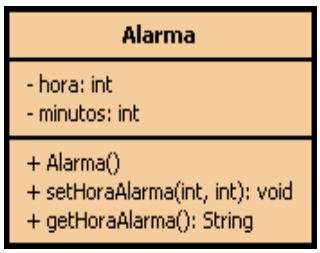
Ejercicios adicionales UT4 (I) – *Interacción de objetos*

Ejercicio 1. Haz una copia (desde el explorador de Windows) del proyecto *Reloj Digital* y dale el nombre *Reloj Digital con Alarma*. A partir de las clases que forman el proyecto *Reloj Digital* vamos a crear un reloj con alarma que incluirá estas dos clases y otras dos más, RelojAlarma y Alarma. La clase RelojAlarma va a modelar un reloj como el que hemos estudiado pero que incorpora una alarma, así cada vez que el reloj avanza si la nueva hora es la misma que la establecida en la alarma se hará sonar un RINGGGGGGGG.

El diagrama de clases es el siguiente:



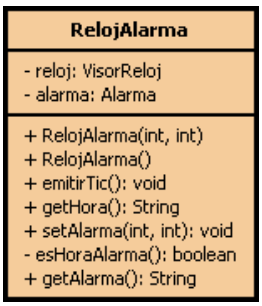




Las clases VisorReloj y VisorNumero no cambian, son las mismas que en el proyecto anterior.

La clase Alarma guarda la hora de alarma, permite modificar esta hora con el método setHoraAlarma() y obtiene la hora de alarma como un String de la forma "XX:XX". El constructor de esta clase establece la alarma, por defecto, a las 7:00 horas.

La clase RelojAlarma tiene dos atributos de tipo referencia, *reloj* de tipo VisorReloj y *alarma* de tipo Alarma. Esta clase incluye:



- un constructor sin parámetros que crea tanto el reloj como la alarma
- un constructor con dos parámetros, la hora y minutos del reloj, que crea el reloj a partir de estos dos parámetros y la alarma
- el método emitirTic() que avanza el reloj. Después de que la hora del reloj avanza se comprueba utilizando el método esHoraAlarma() si la hora actual y la de alarma coinciden. Si es así se hace sonar la alarma emitiendo un "RRRRIIIIINNNNNGGGGG" en la pantalla
- el método setAlarma() permite al reloj cambiar la hora de la alarma
- los métodos getHora() y getAlarma() devuelven la hora actual y la de alarma en formato String
- el método esHoraAlarma() es un método privado. Este método compara dos cadenas, la que representa la hora actual y la que representa la hora de alarma. Para comparar dos cadenas haremos: *cadena1.equals(cadena2)* (La clase String incluye un método equals() que devuelve *true* si las dos cadenas que se comparan son iguales y *false* en otro caso. Consulta la API de Java)

Ejercicio 2. Añade una clase TestReloj dentro del proyecto RelojDigital del ejercicio anterior. Vamos a incluir código en esta clase para probar los métodos de la clase RelojAlarma (lo que hasta ahora hacemos a través de BlueJ interactuando con los objetos desde el *Object Bench* lo vamos a realizar a través de código).

Para ello define:

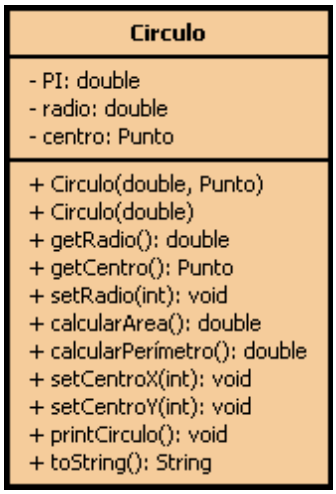
- dos atributos, *miReloj*, *tuReloj* de tipo RelojAlarma
- en el constructor de la clase TestReloj (que no tendrá parámetros) crea *miReloj* con las 18:32 horas y *tuReloj* con la hora 23:58
- define un método test1() en el que:
  - se establece la alarma de *miReloj* a las 18:35
  - visualiza la hora actual de *miReloj* y la hora de alarma
  - se avance el reloj tres minutos
- define un método test2() en el que:
  - se establece la alarma de *tuReloj* a las 7:00
  - visualiza la hora actual de *tuReloj* y la hora de alarma
  - se avance el reloj cinco minutos
  - visualiza la hora actual de *tuReloj*

- Crea un objeto de la clase TestReloj y ejecuta los métodos test1() y test2().

Analiza detalladamente los servicios que proporciona la clase RelojAlarma. Verás que no hay ningún método que nos permita cambiar la hora actual del reloj. Añade un nuevo método a esta clase, el método ponerEnHora() con dos parámetros *hora* y *minutos* que permita el cambio de hora de nuestro reloj. Este método delegará esta tarea en la clase VisorReloj. Prueba este nuevo método en la clase TestReloj incluyendo el método test3().

Ejercicio 3.

- Abre el proyecto *EJAD03 Circulo y Punto*. Analiza el código de la clase Punto. Esta clase no hay que modificarla de momento. Pruébala creando varios objetos Punto y llamando al método toString().
- La clase Circulo modela círculos de un centro y un radio determinados. Se trata de completar el código de esta clase. El diagrama UML muestra los atributos y métodos que incluye la clase.



Relación de composición en UML (un círculo *consta de* un centro que es de tipo Punto)

- Añade a la clase Circulo el atributo *centro*
- Crea los dos constructores de la clase. Son constructores sobrecargados. El primero tiene dos parámetros, el valor del radio y un parámetro de tipo Punto (se pasa un objeto Punto que ya está creado) que será el valor del centro que hay que asignar. El segundo constructor solo tiene un parámetro, el valor del radio. Dentro de este constructor se crea el objeto *centro* con valores iniciales (0,0).

- Prueba ambos constructores creando varios objetos Circulo. Inspecciona cada círculo creado con *Inspect* de BlueJ y observa como se muestra el atributo *centro*.
- Incluye al accesor getCentro() que devuelve el centro del círculo
- Incluye los mutadores setCentroX() y setCentroY(). Estos valores modifican las coordenadas x e y del centro del círculo. Tendrás que llamar dentro de estos métodos a los mutadores setX() y setY() de la clase Punto
- Incluye el método toString() para devolver una cadena con una representación del círculo. Para hacer este método tendrás que llamara a toString() de la clase Punto.
- Incluye el método printCirculo() para visualizar en la pantalla la representación del círculo (llama desde este método al método toString() – es una llamada interna)

Modifica ahora la clase Punto añadiendo los siguientes métodos:

- getDistanciaDesdeOrigen() – devuelve la distancia desde el origen al punto. Se calcula con la fórmula: $\sqrt{x^2 + y^2}$
- getDistanciaDesde(Punto p) – calcula la distancia desde el punto recibido como parámetro con la fórmula: $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$ donde (x_1, y_1) son las coordenadas del objeto actual y (x_2, y_2) las del objeto recibido como parámetro.

Ejercicio 4. Añade al proyecto anterior una clase TestPunto que nos va a permitir testear la clase Punto. La clase TestPunto no va a tener atributos y en el constructor no vamos a incluir ninguna sentencia. Únicamente añadiremos a la clase un método *test()* y dentro de él haremos:

- declarar y crear dos puntos con coordenadas (3, 8) y (12, 9) respectivamente
- visualizar cada uno de los dos puntos en pantalla
- visualizar la distancia del primer punto al origen
- visualizar la distancia que hay entre ambos puntos

Con el método **printf()** podemos formatear la salida en pantalla (algo que no hacen los métodos print() ni println()): (*Consultar aula virtual Moodle*).

System.out.printf(cadena_de_formato, expresiones_a_escribir);

donde:

cadena_de_formato: es un string en el que se incluyen especificadores de formato. Un especificador de formato indica cómo una expresión ha de ser visualizada.
expresiones_a_escribir: puede ser una (o varias) expresiones (s) numérica(s) , o carácter, o booleana (s) o de tipo String

Algunos especificadores usados frecuentemente:

Especificador	Salida
%d	valor entero
%f	valor real
%s	valor String
%c	valor de tipo carácter
%b	valor booleano

Se puede indicar en los especificadores de formato el ancho y la precisión, por ej, %10.2f, significa un ancho para un valor real de 10 posiciones incluido el punto decimal y dos dígitos para los decimales después del punto, con lo que quedan 7 posiciones antes del punto decimal.

Ej.

```
double total = 45.6789;
int contador = 6;
System.out.printf("El valor del total es  %6.2f  y el de contador es %d \n",
                  total, contador);
```

- prueba el método printf() para visualizar la distancia entre los dos puntos

Con **String.format()** podemos hacer algo similar pero devolviendo una cadena formateada en lugar de mostrar resultados por pantalla. (Consulta la ayuda acerca del método estático format() de la clase String).

Ej.

```
String lineaFormateada  = String.format("El valor del total es  %6.2f  y el de contador es %d \n",
                                        total, contador);
System.out.println(lineaFormateada);
```

También podemos formatear números con la clase **DecimalFormat** que está en el paquete *java.text*. (Consultar aula virtual Moodle).