UNIVERSIDAD DE JAÉN

Relación de Ejercicios de E.E.D.D.II

Ingeniería Técnica en Informática de Gestión

La siguiente relación de ejercicios corresponde a la primera parte del curso (clases de objetos y relaciones entre claes) y tiene como objetivo el acercamiento por parte del alumno a ejercicios del tipo que puede encontrar en un examen. La realización de estos ejercicios es de carácter autodidacta y las dudas que surjan pueden preguntarse en tutorías a los profesores de la asignatura.

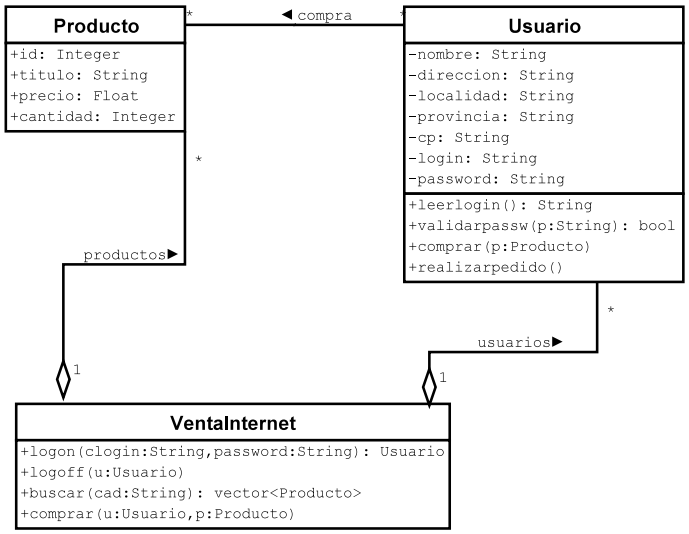
NOTA: CONSIDERAD COMO COMPOSICIONES LAS RELACIONES DE AGREGACIÓN QUE ENCONTREIS.

Ejercicio 1: (disponible solución en web)

El siguiente diagrama UML representa un sistema de venta de libros, CDs y DVDs por Internet, cuyo funcionamiento es el siguiente:

- Un cliente accede a la página web indicando su login y password (operación *logon*). Estos se comprueban en la base de datos de usuarios del sistema y si se corresponden con alguno, se da acceso al sistema.
- A partir de aquí el cliente puede buscar productos indicando una cadena (operación *buscar*). El sistema devolverá el conjunto de productos cuyo título contiene la cadena indicada.
- El cliente podrá comprar los productos que desee, que se irán añadiendo a su compra personal (operación *comprar*). La lista de productos en la compra personal del cliente viene dada por la relación *compra*. El número de unidades del producto existentes en el almacén se actualizará en consecuencia.
- Cuando haya terminado (operación *logoff*) el sistema realizará automáticamente el pedido de todos los productos en su compra personal (operación *realizarpedido*).

Se pide implementar las clases *Usuario* y *VentaInternet* incluyendo las operaciones indicadas (no implementar constructores ni destructores). Usar los contenedores de STL allí donde sea necesario.

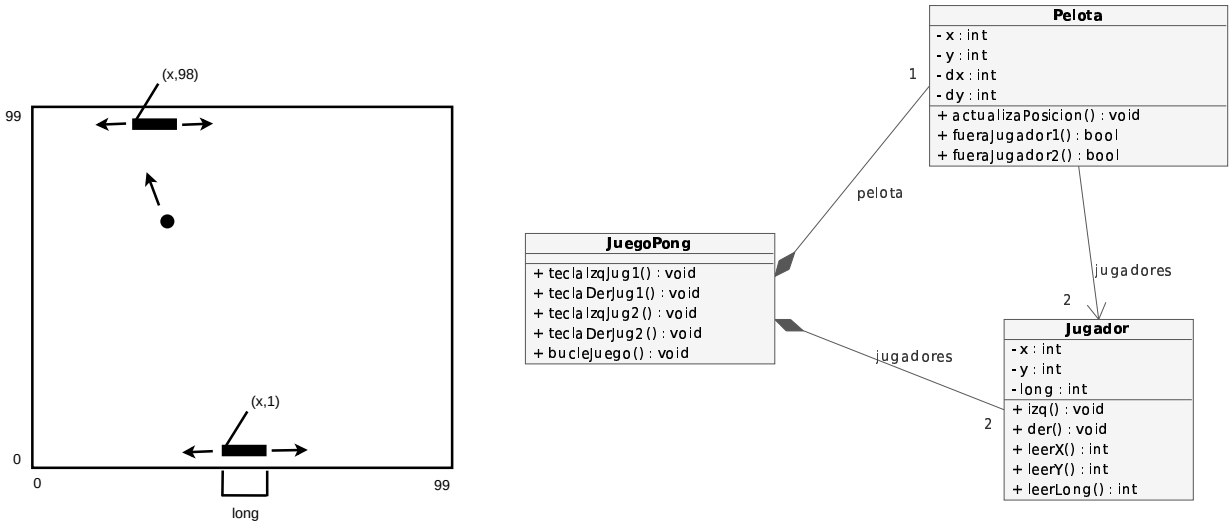


Ejercicio 2:

Uno de los primeros videojuegos de la historia fue el conocido como "Pong". Este juego está inspirado en el tenis: cada jugador dispone de una paleta para enviar una pelota de un lado a otro de la pantalla. Si la pelota supera el lado defendido por uno de los jugadores, éste pierde. Además la pelota puede rebotar en alguno de los dos laterales de la pantalla. El juego puede ser modelado utilizando las clases del diagrama UML adjunto. A continuación se detallan cada una de estas clases:

- *Pelota*. La posición actual de la pelota viene descrita por los atributos x e y. El desplazamiento de la pelota en cada paso viene indicado por dx y dy. El desplazamiento dx puede tener tres valores: *-1* (hacia la izquierda), *0* (derecho) o *+1* (hacia la derecha) y el dy dos valores: *-1* (hacia abajo) y *+1* (hacia arriba). Actualizar el movimiento de la pelota en cada paso es sencillo: basta con sumar dx a x y dy a y. Esta actualización se realiza en la operación *actualizaPosicion()*. Además, en esta operación es necesario comprobar si la pelota ha rebotado en uno de los laterales de la pantalla, en cuyo caso se invierte el desplazamiento dx o en la pala de uno de los jugadores, en cuyo caso se invierte el desplazamiento dy. Las operaciones *fueraJugador1()* y *fueraJugador2()* indican si la pelota ha salido por el lateral de alguno de los dos jugadores (y por tanto ha perdido). Inicialmente la pelota se crea en una posición aleatoria de la pantalla y con un desplazamiento también aleatorio (cumpliendo las restricciones indicadas anteriormente).
- *Jugador*. Esta clase es muy sencilla. Mantiene la posición actual del jugador y la pala de cada jugador. Las operaciones *izq()* y *der()* desplazan la pala a un lado y a otro, evitando que salga de la pantalla.
- *JuegoPong*. Es la clase principal que controla el juego. Las operaciones *tecla*()* mueven la pala del jugador en cuestión a un lado y a otro. Estas operaciones son invocadas directamente desde la interfaz de usuario. La operación *bucleJuego()* controla el funcionamiento del juego, colocando inicialmente cada jugador en el centro de la pantalla, en cada uno de los dos extremos, y creando una pelota aleatoria. A partir de ahí actualiza la posición de la pelota cada medio segundo hasta que sale de la pantalla por alguno de los laterales de los jugadores. En este caso escribe un mensaje indicando el perdedor y reinicia la partida.

El ejercicio consiste en la implementación de estas tres clases, incluyendo los constructores necesarios.



Ejercicio 3:

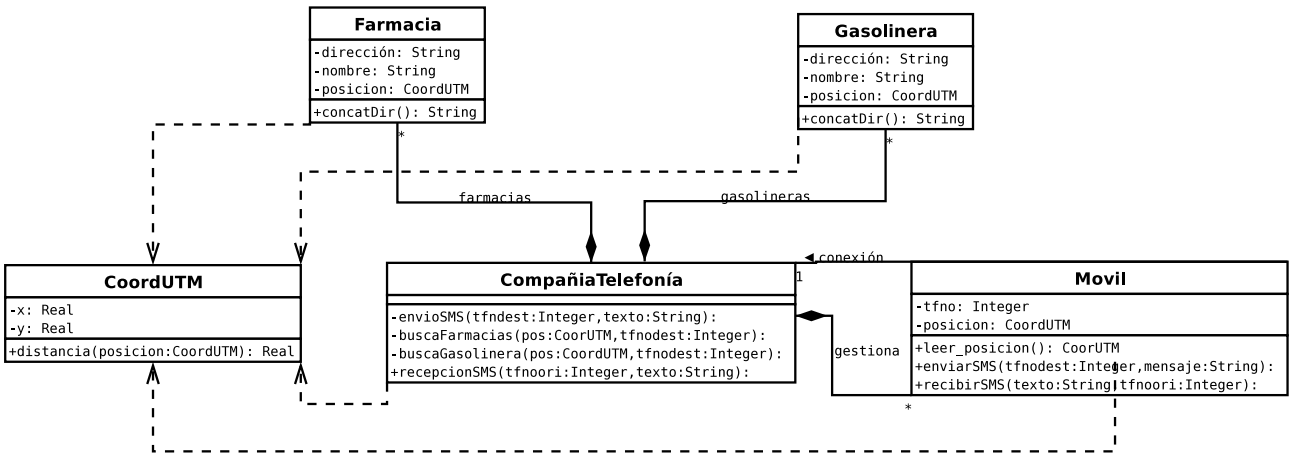
Una compañía de telefonía móvil ofrece, además del sistema de envío de mensajes SMS, un servicio especial a los abonados que tienen incorporado un GPS a su teléfono, de modo que puedan conocer con exactitud las gasolineras o las farmacias más cercanas al lugar donde se encuentran. Concretamente, el sistema porporcionaría al usuario un mensaje SMS indicando la dirección de la gasolinera más cercana y hasta tres farmacias que estén a menos de un kilómetro de donde se encuentre el usuario (por ejemplo las tres primeras que encuentre). Cada móvil actualiza automáticamente sus coordenadas a través de satélite.

El sistema de coordenadas UTM se parece al sistema de coordenadas cartesianas, es decir, puede representarse como (x,y) representando a la latitud y longitud respectivamente con la diferencia de que son coordenadas planas, es decir, puede calcularse la distancia con la fórmula $d(a,b) = \sqrt{(b_x - a_x)^2 + (b_y - a_y))}$.

Un móvil puede enviar mensajes normales a otros teléfonos o realizar una petición mediante un mensaje SMS especial enviado al número 7171 con la clave FARMACIA o la clave GASOLINERA. El sistema le responde así mismo con un nuevo mensaje incluye el texto con las direcciones requeridas.

Cuando un móvil realiza una petición, la compañía telefónica busca las farmacias o las gasolineras más cercanas y envía un SMS al móvil solicitante con la información solicitada utilizando la función privada *envioSMS*.

Las clases Farmacia y Gasolinera cuentan con una función capaz de concatenar la dirección y la posición UTM. Así mismo la clase *CoorUTM* cuenta con una función que calcula la distancia que le separa con otra dirección UTM distinta.



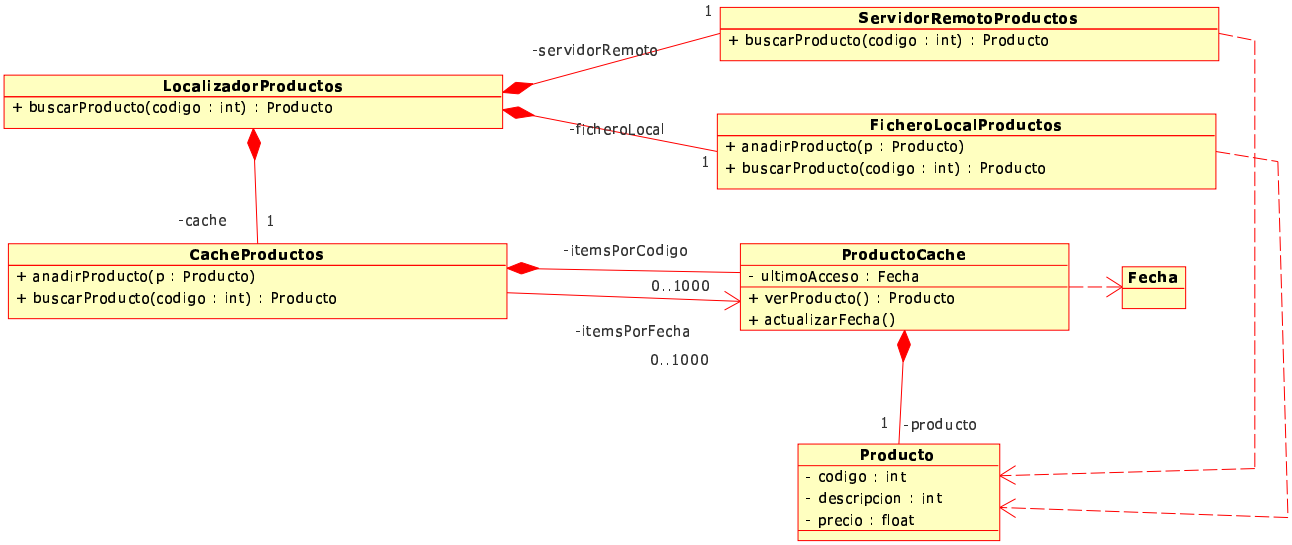
Ejercicio 4:

El siguiente diagrama UML representa un módulo de una aplicación de gestión que proporciona la información de los productos disponibles en un establecimiento. El establecimiento pertenece a una gran cadena donde la información de los productos está centralizada en un servidor. En principio para obtener la información de un producto se realizaría una petición al servidor mediante la operación `buscarProducto` de la clase `ServidorRemotoProductos`, indicando el código del mismo. Sin embargo esta operación es lenta puesto que implica una conexión por Internet que además puede fallar en ocasiones. Por tanto se utilizan varios mecanismos adicionales para acelerar esta operación:

- En primer lugar existe un fichero local, gestionado por la clase `FicheroLocalProductos` donde se almacena una copia de cada producto solicitado a través de `ServidorRemotoProductos`, de manera que la segunda vez que se solicita se dispone de la información localmente, de manera inmediata.
- Para acelerar aún más la búsqueda, los 1000 últimos productos solicitados están disponibles en memoria en una caché, de manera que ni siquiera es necesario realizar su búsqueda en el fichero local. La clase `CacheProductos` contiene 1000 entradas de productos (`ProductoCache`), cada una con la fecha de la última vez que se solicitó. Se dispone de dos mapas, uno ordenado por código para hacer las búsquedas de productos y otro por fecha para saber en todo momento cúal es el utilizado más recientemente.

La clase `LocalizadorProductos` es en última instancia la que se utiliza desde el exterior del módulo. Su operación `buscarProducto()` se encarga de obtener el producto indicado utilizando la fuente más adecuada. Su funcionamiento será el siguiente:

En primer lugar buscará en la caché en memoria. Si el producto existe se actualizará su fecha de último acceso a la fecha actual (operación `actualizarFecha`), se actualizará su posición en el mapa `itemsPorFecha` y se devolverá el producto en cuestión. Si el producto no existe en la caché se buscará en el fichero en disco, se insertará en la caché y se devolverá de manera similar a la anterior. Si ya existían 1000 productos en la caché, entonces se eliminará el más antiguo. Si el producto tampoco existe en disco se pedirá al servidor remoto, se salvará localmente, se insertará en la caché y se procederá de la manera usual. Finalmente, si el producto no existe tampoco en el servidor remoto, se devolverá una excepción indicando que el producto no existe. Las clases a implementar son las siguientes: `LocalizadorProductos`, `CacheProductos`, `ProductoCache` y `Producto`.



Ejercicio 5:

Implementar las clases mostradas en el siguiente diagrama UML, relacionadas con la gestión en tiempo real de un partido de baloncesto:

- *Equipo*: representa cada uno de los dos equipos que disputa el partido. La relación "convocados" contiene la lista de jugadores convocados al partido. Con vistas a facilitar su referencia, cada equipo tendrá asociado un número (1,2), siendo el equipo con código 1 el local.
- *Jugador*: contiene la ficha informativa de cada jugador, con sus datos personales y las estadísticas del partido.
- *ControlPartido*: es la clase fundamental que permite introducir y consultar la información almacenada en el sistema. Sus operaciones son las siguientes:
  - *canasta()*: notifica la consecución de una canasta por un jugador. El número de puntos obtenidos (1,2 o 3) viene indicado por el parámetro puntos. El jugador debe estar en pista (lanzar excepción en caso contrario).
  - *personal()*: notifica la realización de una falta personal por un jugador (lanzar excepción si no está en pista). Además lanza na excepción especial cuando el jugador ya no puede seguir jugando (número de personales == 5).
  - *expulsado()*: indica si un jugador está expulsado por exceso de personales.
  - *cambio()*: notifica un cambio de jugadores. El jugador que sale no puede estar expulsado y debe estar fuera de pista y el que sale debe estar jugando. Lanzar excepciones en caso contrario.
  - *maximoEncestador()*, *numExpulsados()* y *equipoGanador()* permiten obtener las informaciones correspondientes.

Introducir en las clases *Equipo* y *Jugador* todas las operaciones que se consideren necesarias para la correcta implementación del sistema.

Ejercicio 6:

El siguiente diagrama UML representa la estructura de un servicio de telefonía móvil. La clase *Central* representa la central del servicio, con conexión a todos los centros repetidores, es decir las antenas repartidas por la zona donde se presta el servicio (clase *CentroRepetidor*). Dependiendo de su posición, un móvil (clase *Movil*) estará conectado a un único centro repetidor. Vamos a implementar únicamente el servicio de envio de mensajes SMS entre móviles.

El procesamiento de un SMS es el siguiente. En primer lugar se invoca a la operación *enviosms* del móvil, indicando el teléfono destino y el mensaje a transmitir. Éste realiza su transmisión al centro repetidor al que se encuentra conectado (*enviosms*) y éste a su vez pide su procesamiento a la Central (*procesarsms*). En la central es necesario en primer lugar localizar el centro repetidor al que se encuentra conectado el móvil destino del mensaje (interrogar cada centro mediante la operación *conectado*), y una vez localizado, enviar el mensaje al centro (*recepcionsms*) para que éste lo envíe al movil destino (*recepcionsms*). Como el número de móviles conectados a un centro puede ser muy alto, es necesario utilizar una EEDD eficiente para su gestión.

Otro aspecto a contemplar es la conexión/desconexión de un móvil a un centro. Cuando el móvil se desplaza y encuentra un centro que proporciona una señal de mayor potencia que el centro actual, se invoca a la operación *cambiocentro* que desconecta el móvil del centro actual y conecta con el indicado.