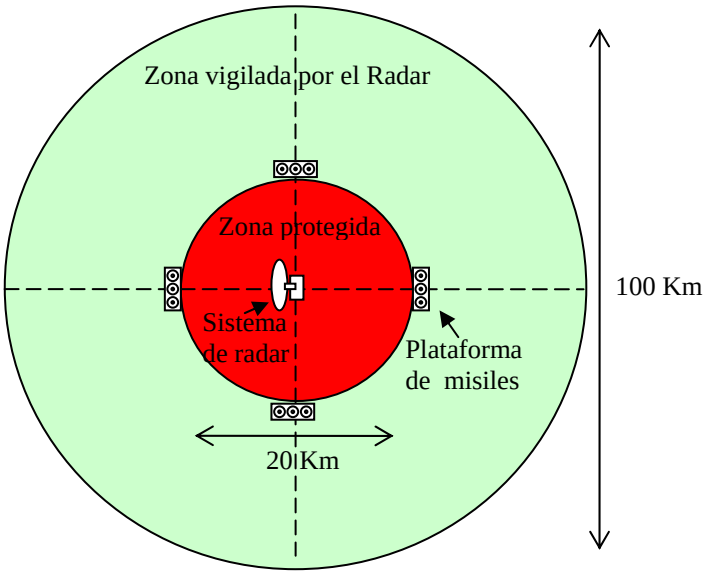
Proyecto SmartHunter

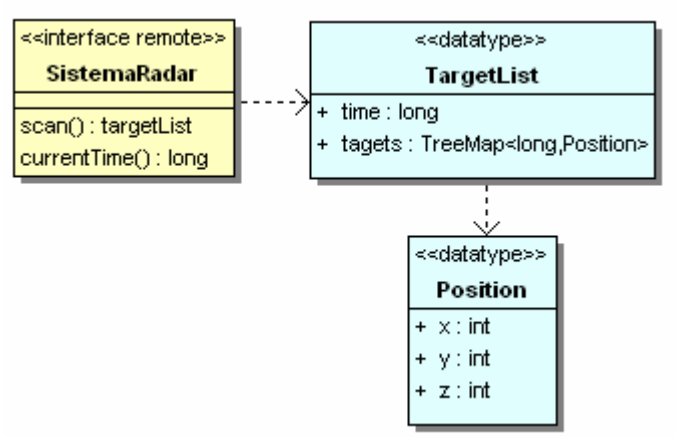
La aplicación consiste en el diseño del software del sistema de defensa de una zona de interés estratégico, basada en misiles inteligentes interceptores y una infraestructura en tierra con un sistema de radar que cubre el espacio aéreo en la región próxima a la zona protegida.

Especificación de la aplicación

El objetivo de la aplicación es defender el espacio aéreo de una zona protegida de interés estratégico que corresponde a un círculo de 10 Km de radio. Su centro se considera como origen de coordenadas [0,0,0]. El sistema de defensa esta constituido por un sistema de radar situado en el centro de la zona protegida. Tiene un alcance de 50Km. Así mismo, en cuatro puntos del perímetro de la zona protegida [0,10Km,0], [10Km,0,0], [0,-10Km,0] y [-10Km,0,0], existen baterías de misiles defensivos inteligentes capaces de recabar información del sistema de radar para localizar su objetivo, y de definir y seguir la trayectoria adecuada para interceptarlo. A partir de su lanzamiento la velocidad del misil es constante e igual a 100 m/s.



Los objetos enemigos se mueven siempre en línea recta con velocidad constante. Su velocidad varía entre 80 y 120 m/s, y la máxima altura que pueden alcanzar es de 5.000 m. Su número es indeterminado, y pueden aparecer por cualquier punto del perímetro cubierto por el radar.
El sistema de radar es un equipo autónomo (que no es objeto del diseño), que ofrece una interfaz remota RMI que ofrece dos operaciones scan() y currentTime(), que respectivamente retorna la lista de objetivos descubiertos y el tiempo actual de su reloj.



Interfaz remota SistemaRadar: La ofrece el equipo (sistema empotrado) que controla el sistema de radar. A través de ella se pueden recabar información de los objetos que se encuentran en la zona de observación. A cada objeto, le asigna un identificador único, que lo identifica de forma unívoca en todas las listas de posiciones que suministre el sistema de radar mientras que permanezca en la zona de observación. Proporciona la localización de cada objeto a través de sus tres coordenadas "x", "y" y "z" con resolución de un metro. El sistema filtra los misiles defensivos, de los que no proporciona información.

Métodos:

- **scan():TargetList** => Retorna la lista de todos los elementos invasores que tiene registrados.
- **currentTime():long** => Retorna el tiempo actual medido por su reloj interno. Este reloj es el que se utiliza para datar los resultados que proporciona.

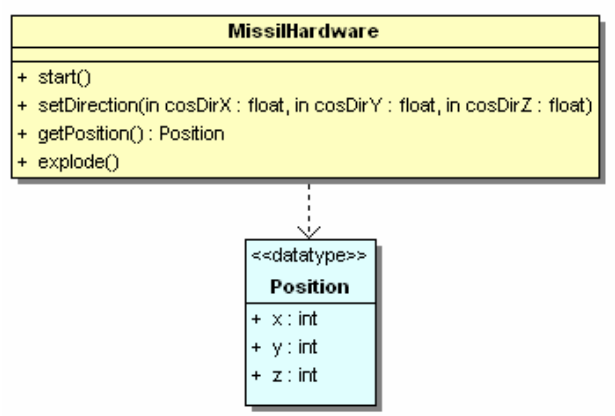
Clase **TargetList**: Define la estructura de datos que retorna el método scan() de la interfaz SistemaRadar. Describe los objetos y la posición que ocupan en el instante concreto en el que se invoca el método.

Atributos:

- **time:long** => Representa el tiempo expresado en milisegundos al que corresponde las posiciones de los objetos descritos en la instancia.
- **targets: TreeMap<long,Position>** => Es una lista diccionario (Map) que contiene una entrada por cada objetivo detectado. La clave es el identificador de cada objetivo y es un long. Entradas con el mismo identificador en diferentes instancias corresponden al mismo objeto. De cada objeto detectado contiene su Posición, caracterizada por las tres coordenadas cartesiana "x", "y" y "z" expresadas en metros.

El misil posee un sistema de conducción autónomo, y un sistema de autoposicionamiento basado en GPS que le permite conocer en todo momento su posición espacial con precisión de 10 m. Así mismo, puede controlar el instante de su explosión.

Los misiles disponen de un sistema de control que controla su operación. Este sistema implementa la interfaz MissilHardware.



Interface **MissilHardware**: Ofrece las operaciones nativas con las que el software puede controlar el misil y conocer su estatus.
Métodos:

start() => Arranca su movimiento con una velocidad constante de 100 m/s

setDirection(cosDirX:float, cosDirY:float, cosDirZ:float) => Establece la dirección de desplazamiento del misil, definida por los cosenos directores en las direcciones X,Y y Z del sistema de referencia fijo en tierra.

getPosition():Position => Retorna la posición actual del misil con una resolución de 10 m.

explode() => Realiza la explosión de la carga destructiva del misil.

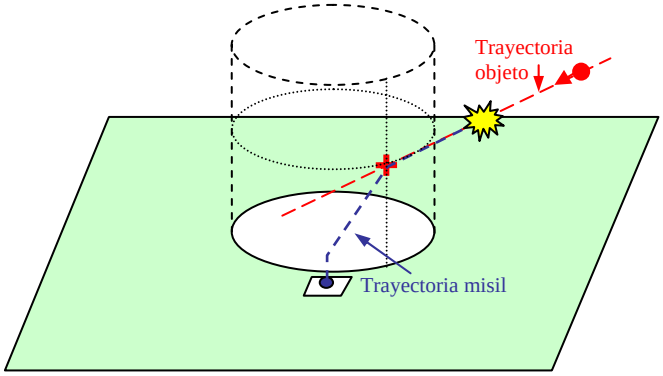
Funcionalidad de la aplicación.

La funcionalidad de la aplicación se formula de acuerdo con un paradigma cliente-servidor.

- **Servidor en tierra**. Implementa las siguientes funciones:
  - Gestiona el sistema de radar y recaba de él información de los objetos que se encuentran en la zona vigilada.
  - Cuando detecta un nuevo objeto, analiza su trayectoria. Si esta no impacta en la zona protegida, la descarta y no le vuelve a prestar atención. Si por el contrario, impacta en la zona protegida, selecciona la batería de misiles más próxima a la zona de intersección de la trayectoria del objeto con el cilindro que cubre el perímetro de la zona protegida y activa en ella un misil defensivo.
  - Mantiene actualizadas las posiciones de los objetos a interceptar y proporciona a los misiles activados para interceptarlos la información que requieran sobre ellos.
  - Visualiza continuamente en un monitor la posición de todos los objetivos (seguidos y descartados) y de los misiles activados para lanzarlos.
  - Considera como interceptado un objeto, cuando transcurre un cierto tiempo sin recibir información de la posición del misil de intercepción. A partir de ese momento descarta la información que reciba del objeto por el sistema de radar (como es simulado, no tiene capacidad de detectar la destrucción del objetivo).

- **Cliente misil.** Tiene asignada la siguiente funcionalidad:
  - Permanece suspendido en espera de ser activado por el sistema de tierra.
  - Una vez activado el misil, se lanza con una trayectoria inicial vertical.
  - Requiere del servidor, cuando lo necesita, la posición del objetivo y continuamente reporta su propia posición.

- Identifica y mantiene la trayectoria del objetivo que debe interceptar.
- Define su trayectoria y controla su movimiento por ella. La trayectoria se compone de tres tramos rectos: Inicialmente avanza verticalmente. Cuando identifica la trayectoria del objeto que debe interceptar, cambia su trayectoria hacia el punto de intersección de la trayectoria del objeto con el cilindro que se sitúa sobre el perímetro de la zona protegida. Cuando alcanza esa posición, cambia de nuevo la dirección de su trayectoria y se dirige directamente hacia el objetivo. Lo alcanza siempre de frente (así lo alcanza aunque el objeto sea más veloz que el misil). Cuando impacta con el objeto, se destruye y deja de operar (deja de reportar su posición al servidor de tierra).



Objetivos

Los objetivos de la práctica son:

- La especificación, el análisis orientado a objetos de los patrones de interacción entre objetos, y del diseño de la aplicación para ser implementado en Java. La herramienta a utilizar para estas actividades es UML.
- El diseño e implementación de una interfaz grafica de usuario utilizando la familia de componentes SWT de Eclipse.
- La documentación que se genera debe ser la adecuada para que un programador que conozca el lenguaje de programación Java, pueda codificar cualquier clase sin que tenga que tomar decisiones de diseño.

Tareas:

Tarea 1: Realizar la especificación de usuario y detallada de la aplicación utilizando diagramas de caso de uso, y diagramas de contexto.

Tarea 2: Realizar el análisis de la aplicación, formulando el diagrama de clases de la aplicación:

- Identificar las clases que constituyen la aplicación, y de cada una de ellas identificar los atributos que constituyen su estado y los métodos que constituyen su interfaz. Establecer entre las clases las relaciones de herencia, agregación, dependencia y visibilidad que existen. Documentar todos los elementos que se incluyan (clases, atributos, métodos y relaciones).

- Describir mediante diagramas de actividad los métodos cuya lógica sea compleja.
- Describir mediante diagramas de estado las clases que tenga un comportamiento complejo.

Tarea 3: Mostrar los patrones de interacción básicos entre objetos. Documentar mediante diagramas de secuencias las interacciones que tengan una complejidad que lo requiera.

Tarea 4: Validar el análisis mediante la comprobación de que todos los casos de uso pueden ser implementados utilizando las clases propuestas.

Tarea 5: Realizar el diseño de la aplicación para que sea implementado utilizando Java:

- Mapear los tipos y los contenedores del análisis a tipos Java.
- Realizar el diseño de concurrencia de las aplicaciones y proponer el framework de de ejecución. Identificar las clases que son activas y los threads que requieren la ejecución.

Tarea 6: Diseñar la interfaz de usuario utilizando la familia de componentes SWT de Eclipse.

Tarea 7: Generar el esqueleto de código Java comprobando la completitud del siseño.

Tarea 8: Recopilar en un breve documento (20 páginas) la información útil para el programador que se ha generado en las anteriores tareas.