# Proyecto
# Serengeti National Park- PARTE 1

## Objetivos:

Practicar de manera más autónoma los conceptos de:

- Modularidad (Clases como TAD's y Singleton)
- Uso de la estructura genérica `ArrayList`
- Excepciones
- Diseño de diagrama de clases UML
- Implementaciones con Java
- Documentación con JavaDoc
- Verificación con JUnit

## Herramientas que vamos a utilizar:

- Herramienta de diseño StarUML
- Entorno de desarrollo Eclipse para Java

## Entregables

Se realizará dos entregas de esta primera parte del proyecto.

- El diseño con StarUML de la aplicación mediante un diagrama de clases para las clases que surjan de los requerimientos expresados más adelante (punto A de las tareas a realizar). Se debe entregar el fichero **.uml** que contiene el diagrama de clases del proyecto con StarUML.
  - Nombre del fichero: *nombre_apellido*.zip
  - La **entrega** es **individual** y se subirá a eGela
  - **Fecha límite** de entrega: **miércoles 1** de **abril** a las **23:55**

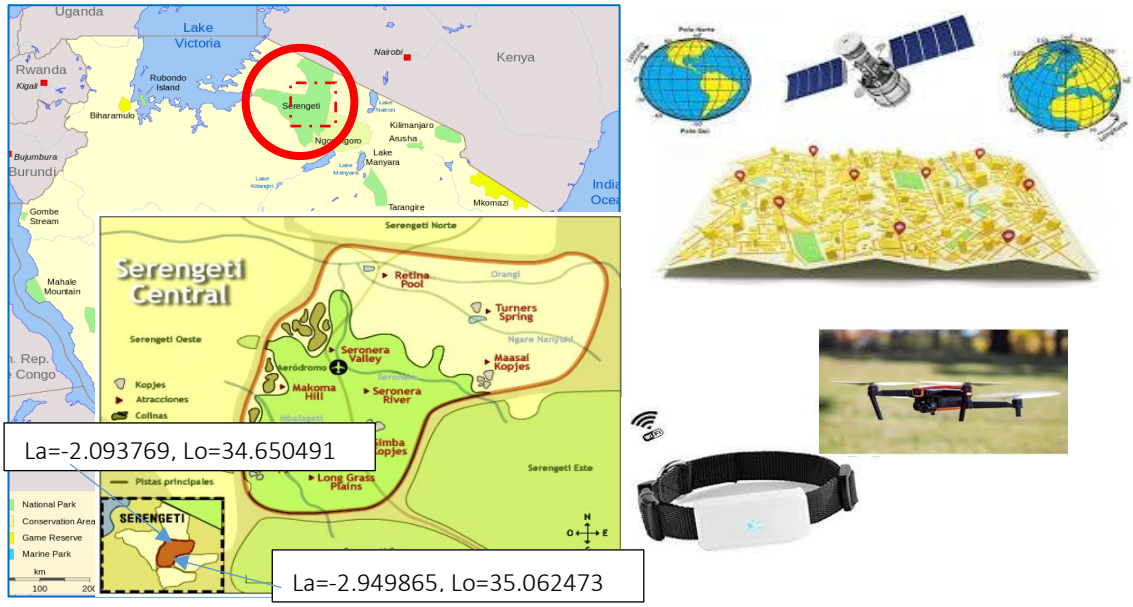
- La implementación de esta primera parte completa, tal y como se indica en las tareas a realizar (puntos B a F). Se exportará el fichero completo que incluirá la carpeta doc de toda la documentación del proyecto.
  - Nombre del fichero: *nombre_apellido*.zip
  - La **entrega** es **individual** y se subirá a eGela
  - **Fecha límite** de entrega: **domingo 26** de **abril** a las **23:55**

## Visión general de la aplicación

El **Serengeti** está localizado en el norte de Tanzania (África). El nombre Serengeti deriva del idioma Maasai, Maa; específicamente, "Serengit" significa "Llanuras infinitas". El parque nacional del Serengeti tiene como misión la conservación de la naturaleza y por ello desea monitorizar la fauna de un determinado territorio. La idea es capturar ejemplares de interés y soltarlos con una unidad de rastreo. La unidad de rastreo registra las coordenadas GPS de la localización del ejemplar. Se dispondrá de una serie de drones para poder visualizar, a través de capturas de imágenes, determinados puntos GPS y ejemplares.

Se quiere implementar una aplicación para controlar y gestionar los datos recopilados desde las unidades de rastreo en una zona concreta del parque. En el diseño de la aplicación

deben aparecer en el packge `serengetiPark` las clases SerengetiMonitor, TrackingDevice




(Unidad de rastreo), GPS, Drone y Specimen con la representación y comportamiento que se describe a continuación. Sobrescribe en todas las clases el método `toString` para que devuelva la información de la clase como un String.

**C1.** La clase **GPS** representa las coordenadas de un punto en la superficie terrestre.

a. Se caracteriza por su latitud y longitud.
b. Tendrá una única constructora con dos parámetros latitud y longitud.
c. Un método `distanceTo`[1] devolverá *la distancia (aproximada) hasta otro punto GPS dado.*
d. Son necesarios los getters sobre latitud y longitud.

**C2. Drone** representa a los drones que se utilizan para monitorizar el parque.

a. Está caracterizado por un nombre, su disponibilidad y la posición GPS en la que se encuentra.
b. Se creará con una posición GPS y un nombre dados. Inicialmente estará disponible.
c. Debemos ser capaces de conocer y modificar la disponibilidad del dron.
d. Debemos poder conocer la posición del Drone (`whereIAm`).
e. El método `moveAndCaptureImage` tendrá como objetivo capturar una imagen en una posición GPS dada y devolverla. Una imagen será una matriz de bytes de 1000*1000 (`byte[][]`[2]). Debido a la distribución de postes de carga en puntos estratégicos del parque los drones tienen una autonomía ilimitada.
   - Para la implementación de este método se deben usar los siguientes métodos de la clase (los tenéis en eGela):

[1] Utiliza como aproximación el teorema de Pitágoras y multiplícalo por un factor corrector de 85 (serán los kms aproximados). Tienes las operaciones **`Math.pow(a, b)`** para expresar **a** elevado a **b**, y **`Math.sqrt(a)`** que expresa la raíz cuadrada de **a**.

[2] byte[][] es un array de arrays (new byte[1000][1000]- en este proyecto NO se va a trabajar diretamente con las matrices para manipularlas ni acceder a sus valores a[i][j].

- El método `move`, simula el desplazamiento del dron a una posición dada (por ello en su ejecución se simulación realmente el desplazamiento y tarda un tiempo su ejecución):

```
public void move(GPS newPosition);
```

- El método `tryToCapture`, simula el intento de captura de la imagen (recuerda, una imagen será una matriz de byte). Los drones debido a las inclemencias del tiempo, sus vibraciones u otros aspectos, pueden fallar en su intento de capturar una imagen elevando en esos casos la excepción `CaptureErrorException`:

```
private byte[][] tryToCapture() throws CaptureErrorException;
```

  - En este método el dron que debe estar disponible, pasará a no disponible, se desplaza a la posición indicada (si es que no está ya en dicha posición), intenta capturar la imagen y en caso positivo vuelve a la posición original en la que estaba, pasa de nuevo a estado disponible y devuelve la imagen. El método moveAndCaptureImage seguirá elevando la excepción `CaptureErrorException` si no ha podido obtener la imagen y no volverá a la localización original.

**C3.** El **`TrackingDevice`** es el dispositivo que tienen los ejemplares que se monitorizan y se caracteriza por la última posición GPS obtenida. La funcionalidad implementada (`whereIAm`) simula el movimiento del ejemplar en un periodo de tiempo (30 minutos), creando y registrando la nueva localización GPS y devolviéndola. Esta clase se da implementada (está en eGela).

**C4.** Un **`Specimen`** representa a cualquier ejemplar monitorizado del parque.

a. Se caracteriza por su nombre, su recorrido del día, la localización GPS al inicio del día, y el dispositivo de rastreo asociado. Usa **`ArrayList`** para representar el recorrido del ejemplar.
b. La constructora deberá incluir información del nombre, la posición GPS inicial de localización y será la responsable de crear en esa posición inicial y guardar el dispositivo de rastreo asociado.
c. Además, debe existir la posibilidad de:
   - Obtener la última posición conocida del ejemplar actual (`lastPosition`). Se devolverá la última posición registrada. Si no hay registros, la última posición será la posición de inicio del ejemplar.
   - Consultar la distancia recorrida hasta ese momento (`kmsTraveled`).
   - Devolver el nombre del ejemplar (`getName`)
   - Registrar y devolver la posición actual del ejemplar (`register`).
   - Simular el movimiento del ejemplar durante el tiempo dado (int, son minutos) que puede suponer el registro de varias posiciones (`updatePositions`). El método registra tantas posiciones GPS según periodos de tiempo de 30 min aprox. Este método hará uso del método `whereIAm` del dispositivo de rastreo asociado al ejemplar actual.

     Por ejemplo, si el tiempo que se va a simular es de 180 minutos entonces se realizarán 6 registros de posiciones para el ejemplar (6*30=180). Si el valor del tiempo es menor que 30 se realiza al menos un registro y si el valor es mayor que

30 pero no múltiplo de 30, siempre se realiza un registro más. Si fueran 157 (5*30+7) serían 6 registros

**C5.** Finalmente, el **`SerengetiMonitor`** incluye toda la información necesaria para la monitorización de ejemplares. Lo representaremos como un `Singleton`, ya que la información relativa a la monitorización debe estar centralizada. Las características y funcionalidades deseadas son:

a. Las listas de los ejemplares monitorizados y los drones caracterizan la monitorización del parque. Ambas se implementarán con la estructura **`ArrayList`**.
b. Añadir un nuevo ejemplar a monitorizar (`addSpecimen`).
c. Añadir un nuevo dron (`addDrone`)
d. Obtener la lista de ejemplares monitorizados que están situados (en su última posición conocida) a menos de **max** kilómetros (valor dado) de un determinado ejemplar dado (`closestSpecimen`). En esa lista no debe estar el ejemplar dado.
e. Como los drones pueden fallar en su intento de capturar una imagen se ha establecido un proceso de captura de imagen que dado un número **n** de intentos posibles y un ejemplar realiza la captura de la siguiente manera (`captureWithAttemps`): (a) Se busca un dron disponible y se manda a capturar una imagen en la posición del ejemplar dado; (b) Se reintentará la captura de la imagen hasta un máximo de **n** veces (valor dado); (c) En el momento que se realiza la captura de imagen se devuelve; (d) Si a pesar de todos los **n** intentos no se puede completar la operación se enviará al dron a su posición original, se indicará que está disponible y se seguirá intentando con otro dron disponible; (e) Si finalmente no es posible capturar la imagen con ningún dron disponible, el proceso elevará la excepción (`ImpossibleToCaptureException`).
f. Además, se desea recopilar imágenes (`collectImage`**`)`** de los ejemplares que están a menos distancia de un número de kilómetros dado de un ejemplar dado. Esto es muy útil por ejemplo si existe un depredador importante y deseamos obtener las imágenes de todos los ejemplares cercanos. En este tipo de recopilación se debe intentar 5 veces la captura de imagen por ejemplar y si no fuera posible se escribirá el nombre del ejemplar y un mensaje del error de captura de la imagen. Un error de captura en un ejemplar **no debe impedir** la captura de la imagen de los demás. Al final deberá mostrar un mensaje indicando el número de ejemplares de los que no se ha podido obtener una imagen (si es que hay alguno) y devolver la lista de imágenes capturadas[3].
g. En esta clase se permite simular el paso del tiempo y con ello el movimiento de los ejemplares en el parque. Dado un valor del tiempo que se va a simular (`int`, en minutos), solicita el registro de las posiciones de los ejemplares durante ese tiempo (`updatePositions`).

---

**Tareas a realizar:**

A. Diseña con StarUML la aplicación mediante un diagrama de clases para las clases que hayan surgido de los requerimientos expresados anteriormente.

B. Una vez completado todo el diagrama obtén el esqueleto Java de la aplicación. Implementa y documenta el esqueleto Java obtenido del diseño realizado.

C. Implementa de manera externa las excepciones en un `package` llamado **`exceptions`**.

D. (opcional) Como ayuda transitoria puede venir bien crea para cada clase un `main` para crear pequeñas pruebas de funcionamiento.

[3] Observar que se devolverá un ArrayList<byte[][]>. No hay ninguna dificultad ya que los elementos se tratan como hasta ahora; byte[][] es como cualquier otro tipo de dato referenciado.

Ejemplo de pruebas de funcionamiento para la clase GPS (no son necesarias tantas):

```
La distancia entre bernabeu y Camp Nou es (40.452961, -3.688333) y
(41.380833, 2.122778) es 500.2014023941193
La distancia entre dos puntos cercanos es (43.5372001, -5.6370439) y
(43.5402668, -5.6475824) es 0.9329292363231048
La distancia entre dos puntos muy cercanos es (43.5291675, -5.639102) y
(43.5292752, -5.6390161) es 0.011709687122280063
La distancia entre los puntos (42.8591338, -2.6818614) y (43.318334, -
1.9812313) es 71.2048079914636
La distancia entre los puntos (42.8591338, -2.6818614) y (43.2630126, -
2.9349852) es 40.51476144240195
La distancia entre los puntos (43.2630126, -2.9349852) y (43.318334, -
1.9812313) es 81.2053432923684
```

E. Diseña e implementa[4] con JUnit en un package **test**, todas las pruebas necesarias mediante estudio de caja negra (clases de equivalencia y casos límite si son necesarios) sólo para las siguientes clases y métodos:
   1. **GPS**: distanceTo. En realidad, esté método se probaría con una única prueba ya que tiene un único comportamiento posible, sin embargo, prestaremos atención a dos casos de prueba: distancia entre el mismo punto y distinto punto. Debido a las aproximaciones del cálculo se puede hacer uso del **equals** con una cierta aproximación:

      **assertEquals**(double expected, double actual, double delta)
      Asserts that two doubles are equal concerning a delta.

   2. **Specimen**: kmsTraveled. Casos a considerar: sin puntos registrados de desplazamiento, con un punto registrado, con varios puntos registrados.
   3. **SerengetiMonitor**[5]: closestSpecimen, captureWithAttemps y collectImage.

F. Finalmente, crea una clase **SerengetiSimulator** en el paquete packSimulator y en su **main** debe realizar lo siguiente:
   1. Generar y obtener el objeto que representa la monitorización del parque.
   2. Crear 10 ejemplares de diferentes especies en posiciones cercanas y añádelos al parque. Tenéis un fichero posiciones.txt con posiciones del parque cercanas entre sí. Cada línea contiene un par de coordenadas latitud longitud separadas por un espacio en blanco.
   3. Simular que ya han pasado 6 horas de registros de movimientos.
   4. Obtener todos los ejemplares cercanos a una distancia de 300Km del primero de los ejemplares. Imprimir todos esos ejemplares informando de su nombre, posición y distancia al primero.
   5. Crea 3 drones en la misma posición GPS que quieras (por ejemplo, la=-2.949865-lo=35.062473) e incorpóralos al parque. Indica que el primer dron no está disponible.
   6. Recolecta imágenes de los animales próximos en 90 Km al primero de los ejemplares creado e indica cuántas imágenes se han podido obtener.

[4] Quizás para hacer correctamente las pruebas y poder volver a estados conocidos, sea necesario añadir alguna operación más en alguna clase no considerada inicialmente.

[5] Observad que el Singleton se refiere a un único objeto, es decir, una vez creado se manipulará el mismo en todos los test. Por tanto, tenemos que asegurar que las modificaciones sobre el Singleton en una prueba puedan deshacerse de algún modo para empezar desde un estado conocido en la siguiente prueba.

Captura una ejecución que permita visualizar adecuadamente el funcionamiento de la monitorización del parque. Una posible traza de ejecución a través de la pantalla (`Console`) podría ser la que se muestra en la siguiente página.

**NOTA IMPORTANTE: como se está haciendo una simulación de lo que sucede en el parque, la ejecución tarda un rato. Tened paciencia. Además, cada ejecución da resultados distintos, ya que la simulación de los movimientos de los animales y de la obtención o no de cada imagen es aleatoria.**

```
Animales cercanos a menos de 300 Km del primero ...
s2 (-3.1187810514855183, 34.89355694851448) distancia 54.17983521841921
s3 (-2.8817850404187806, 36.286838503669) distancia 156.51240446760437
s4 (-3.734726216889369, 35.12968715806584) distancia 108.24616536581894
s5 (-2.701970084566289, 34.81123371694908) distancia 30.163670950888736
s6 (-3.297759915433711, 34.880956150727826) distancia 65.55474231885707
s7 (-2.702806176470588, 35.30953182352941) distancia 72.34442431005019
s8 (-4.238100294117645, 34.646498124985065) distancia 135.7842194420704
s9 (-2.149865, 35.10785963972081) distancia 69.65191753773907
s10 (-3.614570882352941, 33.97590988969095) distancia 91.61429831876856


Especie en revisión s1 (-2.651566893419678, 34.4599647242871)
  captura de imágenes de animales cercanos ... a 90Km
==> Revisión s2 (-3.1187810514855183, 34.89355694851448)

  --intento 1 d1-true -2.949865 35.062473
  ...desplazándose a -3.1187810514855183 34.89355694851448 ...    ... he llegado.
    ... intento de realizar la captura ...116... imagen capturada ...
  ...desplazándose a -2.949865 35.062473 ...    ... he llegado.
==> Revisión s5 (-2.701970084566289, 34.81123371694908)

  --intento 1 d1-true -2.949865 35.062473
  ...desplazándose a -2.701970084566289 34.81123371694908 ...    ... he llegado.
    ... intento de realizar la captura ...26... imagen capturada ...
  ...desplazándose a -2.949865 35.062473 ...    ... he llegado.
==> Revisión s6 (-3.297759915433711, 34.880956150727826)

  --intento 1 d1-true -2.949865 35.062473
  ...desplazándose a -3.297759915433711 34.880956150727826 ...    ... he llegado.
    ... intento de realizar la captura ...105
  --intento 2 d1-false -3.297759915433711 34.880956150727826
    ... intento de realizar la captura ...89... imagen capturada ...
  ...desplazándose a -3.297759915433711 34.880956150727826 ...    ... he llegado.
==> Revisión s7 (-2.702806176470588, 35.30953182352941)

  --intento 1 d1-true -3.297759915433711 34.880956150727826
  ...desplazándose a -2.702806176470588 35.30953182352941 ...    ... he llegado.
    ... intento de realizar la captura ...55... imagen capturada ...
  ...desplazándose a -3.297759915433711 34.880956150727826 ...    ... he llegado.
==> Revisión s9 (-2.149865, 35.10785963972081)

  --intento 1 d1-true -3.297759915433711 34.880956150727826
  ...desplazándose a -2.149865 35.10785963972081 ...    ... he llegado.
    ... intento de realizar la captura ...33
  --intento 2 d1-false -2.149865 35.10785963972081
    ... intento de realizar la captura ...77... imagen capturada ...
  ...desplazándose a -2.149865 35.10785963972081 ...    ... he llegado.

... se han capturado 5 imagen/es.
... había 5 animales a menos de 90 Km.
```