Examen final Prog. III

- Antes de empezar indica tu posición en el aula en la url: http://bit.ly/ud-iniexamen
- Puedes utilizar el programa de control durante **todo el examen** y en ese caso tendrás acceso a internet. En caso contrario, solo se permite la conexión durante cuarenta minutos y a **partir de ese momento debes desconectar wifi/red. Está prohibido cualquier flujo de información relativo al examen.**
- En la web de la asignatura ("Entregables y Evaluación") tienes los ficheros con los que se va a trabajar en un archivo comprimido. Crea un proyecto en eclipse y descomprime ese fichero en el proyecto.
- Para entregar el examen, comprime todo el directorio src, y renombra el fichero (.zip o .rar) con tu dni, nombre y apellido. Asegúrate de que incluye todos los ficheros que has modificado.
- Al acabar el examen, vuelve a conectarte y entrega por ALUD en la tarea correspondiente.
- Es imprescindible también rellenar la encuesta de final de examen: http://bit.ly/ud-finexamen (Especialmente importante para describir qué has realizado en la última tarea).
- No se contestarán a preguntas o dudas sobre el examen, solo a problemas graves con los equipos. Para cualquier duda sobre el enunciado utilizar el documento compartido http://bit.ly/ud-dudaexamen
- Entrega en 2h:30'=sin limitación de nota. Entrega en 4h:30'=limitación a 8 (+ sobrante de 10 si se supera).

1. Planteamiento

Para este examen, vamos a partir de un sistema de visualización de zonas ajardinadas geolocalizadas inicializado con datos de Erandio. Ejecuta el fichero principal `EdicionZonasGPS.java`. Observa que ves las zonas cerradas, árboles situados en esas zonas, y puedes moverte y hacer zoom por la pantalla.

La base del sistema son las siguientes clases ya existentes:

- **`ItemEnMapa`**. Permite definir cualquier punto geolocalizado en el mapa. Clase abstracta.
- **`Arbol`**. Hereda de ItemEnMapa y permite definir árboles con nombre, geolocalización y color.
- **`Zona`**. Permite definir toda la información de puntos GPS de cada zona ajardinada.
- **`GrupoZonas`**. Agrupa una serie de zonas de trabajo. Ya creado un grupo para este examen: `jardinesErandio`. También está creada la lista de árboles que es un arraylist independiente: `arbolesErandio`.
- **`EdicionZonasGPS`**. La clase principal con la ventana.
- **`UtilsGPS`**. Un par de utilidades que necesitarás en algunas tareas.

2. Tareas

Se pide que realices las siguientes tareas. Puedes elegir las que veas conveniente, no hay que hacer todas: cada una indica su puntuación (sobre 10) y su tamaño en líneas de código aproximadas.

Tarea 1 [2]. JUnit (*UtilsGPSTest.java – unas 30 líneas*). Vamos a comprobar que el método `gpsDentroDePoligono` de la clase *UtilsGPS* es correcto. Este método comprueba si un punto dado está o no dentro de un polígono de puntos. Para asegurarlo prueba las dos siguientes cosas:

- Todos los árboles de Erandio (GrupoZonas.arbolesErandio) están dentro de alguna de las zonas ajardinadas de Erandio (GrupoZonas.jardinesErandio).
- Ningún punto en los extremos del mapa está dentro de las zonas ajardinadas de Erandio. Puedes suponer los siguientes extremos del mapa: (43.323, -2.990) y (43.287, -2.923). Calcula 100 puntos en cada uno de los lados de ese rectángulo y comprueba que ninguno de esos 400 puntos está en una zona ajardinada.

Tarea 2 [2,5]. EEDD (*GrupoZonas.java – unas 10 líneas*). La clase *GrupoZonas* utiliza un `TreeSet` para almacenar las zonas del grupo: las queremos en orden de código y una misma zona no puede repetirse. Sin embargo esto hace que para buscar la zona después dado su código (método `getZona`) la eficiencia sea muy mejorable.
Modifica la estructura de datos interna `setZonas` **cambiándola por la que decidas**, para mejorar la eficiencia del método `getZona`, manteniendo el orden de zona. Observa que además del método indicado necesitarás cambiar también algunos métodos más para que todo siga funcionando. Indica en los comentarios del método cuál es la mejora que crees que se produce con tus cambios en tiempo de ejecución. La eficiencia se valora en la evaluación.
(Ten cuidado porque si no completas esta tarea puede dejar de funcionar el resto del código. Recomendable que dupliques la clase por si acaso)

Tarea 3 [3]. Componentes visuales (*EdicionZonasGPS.java, 55 líneas*). [2 puntos] Añade una *JTable* en la zona izquierda de la ventana que visualice los datos de los puntos de una zona, como se muestra por ejemplo en la figura lateral. Cuando se seleccione una zona en la lista derecha de zonas de la ventana, en la tabla deben aparecer todos los puntos de esa zona, con la indicación del código de zona, el área (número secuencial de subzona), el punto (número secuencial de punto dentro de la subzona), y la latitud y longitud.
Ten en cuenta que el último punto de cada subzona es el mismo que el primero (el cierre del área), y ese NO debe incluirse en la tabla.
La tabla debe tener las siguientes anchuras de columna, sucesivamente (en píxels): 40, 30, 30, 120, 120 (310 píxels de anchura total la tabla).

[1 punto] Permite además que si se edita en la tabla interactivamente una latitud o longitud, se cambie en la zona correspondiente el punto visual y se dibuje en pantalla el punto cambiado.
Tienes en el código varios comentarios que indican los puntos donde tienes que editar el código. También puedes consultar en ALUD un vídeo que ilustra lo que ocurre al codificar bien esta tarea.

Edición de zonas geoposicionadas (Erandio)

Zona 1 tiene 3 subzonas y 26 puntos diferentes.

Zona	Área	Pto	Latitud	Longitud
01	0	0	43.30321371568...	-2.97551574566...
01	0	1	43.30324897365...	-2.97532906306...
01	0	2	43.30261103851...	-2.97510554313...
01	0	3	43.30257812165...	-2.97529012698...
01	1	4	43.30231888722...	-2.97519569212...
01	1	5	43.30250455548...	-2.97428388921...
01	1	6	43.30220791852...	-2.97420880810...
01	1	7	43.30201904526...	-2.97507535488...
01	2	8	43.30635760991...	-2.97321652000...
01	2	9	43.30645708539...	-2.97272783300...
01	2	10	43.30643808397...	-2.97272069002...
01	2	11	43.30645121503...	-2.97265508782...
01	2	12	43.30646652424...	-2.97266087613...
01	2	13	43.30658254463...	-2.97208525216...
01	2	14	43.30656633487...	-2.97207919930...
01	2	15	43.30657739257...	-2.97200572497...
01	2	16	43.30619286067...	-2.97186225596...
01	2	17	43.30617712200...	-2.97194117573...
01	2	18	43.30613714688...	-2.97192623157...
01	2	19	43.30611883565...	-2.97201295861...
01	2	20	43.30612662449...	-2.97201604248...
01	2	21	43.30605135087...	-2.97240681645...
01	2	22	43.30604191141...	-2.97240340653...
01	2	23	43.30602543788...	-2.97249104241...
01	2	24	43.30606261323...	-2.97250530632...
01	2	25	43.30594660577...	-2.97307219245...

Tarea 4 [2]. Base de Datos (*Tareas.java, 35 líneas, además del código clásico de base de datos*). Queremos poder guardar en base de datos los árboles de cada zona. Para ello ya hay un botón en la ventana (Tarea 4) que llama al método `Tareas.tarea4()` que no hace nada. Prográmalo para que al pulsarlo se actualicen en base de datos los árboles que hay en la zona seleccionada en la lista lateral de la ventana (si no hay ninguna, no se hace nada).
Tienes ya la clase `BD.java` programada con la mayor parte del código necesario, para insertar y buscar filas en la tabla arbol (formada por campos zona string, latitud double, longitud double, nombre string, edad int, color int). Probablemente tendrás que añadir algún método.
El objetivo es conseguir que al pulsar el botón, todos los árboles de la zona seleccionada se sincronicen con la base de datos de modo que queden los mismos en ambos lugares (manteniéndose intactos los de otras zonas). Ten en cuenta que un árbol se puede mover interactivamente en la ventana (drag sobre cualquier árbol con el botón de "Mover árbol" pulsado), y que al actualizarse la base de datos puede haber árboles que antes no estaban en la zona y ahora están, o viceversa.
Te será útil el método `UtilsGPS.gpsDentroDePoligono` ya programado.

Tarea 5 [2,5]. Recursividad (*Tareas.java – 30 líneas*). Ya hay definido un botón "Tarea 5" que de momento solo recoloca la posición del mapa para que se vean todas las zonas. Modifica el método **calculoCentroides** de la clase `Tareas` para realizar lo siguiente:
Se han definido tres puntos centrales (centroides) aleatorios en el mapa. Queremos trabajar con los árboles en tres grupos, y vamos a definir un proceso estadístico para hacerlo, basado en el llamado *k-means* que se realiza recursivamente de la siguiente manera:

1. Se recorren todos los árboles y se les asigna un color dependiendo del centroide más cercano: rojo, azul, verde (respectivamente para el primer, segundo y tercer centroide). Así, cada árbol toma el color correspondiente a su centroide más cercano. (Distancia entre 2 puntos ya programada en la clase *UtilsGPS*)
2. Se dibuja la pantalla con los árboles recién coloreados (paso ya codificado).
3. Se dibujan los centroides (ya codificado).
4. Se hace una pausa de un par de segundos (ya codificado).
5. Se calculan las medias de posición de cada uno de los grupos de árboles: media de sus coordenadas de longitud, media de sus latitudes (los árboles del mismo grupo son los que tienen el mismo color).
6. Se mueven los centroides a esas coordenadas medias (cada centroide se mueve de acuerdo a su grupo).
7. Se realiza el proceso recursivamente, hasta el caso base: que en la asignación de colores del paso 1 *ningún árbol cambie de color* con respecto al que tenía.

Puedes consultar en ALUD un vídeo que ilustra lo que ocurre al codificar bien esta tarea.

Tarea 6 [hasta 1 punto]. Libre. Añade a la aplicación cualquier mejora que desees. Es imprescindible indicar con comentarios `// TAREA 6` los sitios de código que modifiques, y que indiques en el formulario de entrega una pequeña descripción de lo que has realizado. Se valorará en función de la calidad, originalidad y complejidad.