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 el lugar del código que hay que editar.

Tarea 1 [2]. JUnit (*UtilsGPSTest.java*). Hay dos métodos en la clase *UtilsGPS* que queremos probar, y a la vez comprobar la corrección de todas las zonas definidas. Haz las siguientes pruebas:

- El área de todas las zonas de Erandio es mayor que cero (utilizando el método `areaDePoligono`).
- La intersección entre las áreas internas de cada zona es cero (utilizando el método `interseccionEntrePoligonos`).

Es probable que al hacer estas pruebas descubras que hay un par de zonas incorrectas (que no cumplen alguna de estas condiciones). Localízalas e indícalas en los comentarios del método (hay un hueco preparado para ello). Puedes comprobar visualmente que estos errores efectivamente ocurren.

Tarea 2 [2,5]. EEDD (*Tareas.java*). El ayuntamiento quiere hacer un conteo de árboles de acuerdo a su especie (el *nombre* del objeto árbol). Para ello se pide que a pulsar el botón (*tarea 2*) recorras todos los árboles, compruebes cada árbol en qué zona está, y lo introduzcas en una estructura de datos en la que ir contando los árboles que en cada zona son de la misma especie. Para ello **debes utilizar** la siguiente estructura de datos:

```
TreeMap<String,TreeMap<String,Integer>> conteoArboles
```

El mapa general es de clave código de zona (*String*), y en cada zona con árboles debe guardarse otro mapa en el que se asocie a cada especie de árbol (clave *String*) su número de conteo (valor *Integer*). Genera primero el mapa completo, y luego recórrelo para visualizar por consola el listado de árboles, ordenado por zona y por nombre de especie. El formato deberá tener una forma similar a esta:

Zona 01 - 1 árbol(es) Morera japonesa
Zona 01 - 1 árbol(es) Nogal
Zona 02 - 1 árbol(es) Abedul
...
Zona 77 - 2 árbol(es) Roble
Zona 77 - 1 árbol(es) Álamo
Zona 80 - 1 árbol(es) Álamo

. ----------

Tarea 3 [3]. **Componentes visuales** (*EdicionZonasGPS.java*). [1 punto] Añade un evento a la *JTable* existente para que cada vez que se selecciona una fila en la tabla se dibuje en la zona correspondiente el punto que se ha seleccionado. Observa que en la propia tabla tienes la información de la zona, el número de subzona y el número de punto a dibujar. Hay un método *dibujaCirculoNegro* que puedes usar para el dibujado.

[1 punto] Añade código de render a la tabla para que las filas que corresponden a una subzona (área) par se dibujen con un fondo blanco y las subzonas impares con fondo gris claro (diferenciando visualmente de esa manera las distintas subzonas dentro de la zona). **Debes localizar dónde hacer esta tarea.**

[1 punto] Añade código de gestión de eventos a la tabla para que al pulsar la tecla Supr (Del) se borre la fila seleccionada, y se borre igualmente el punto del gráfico correspondiente a esa fila. Puedes llamar al método `calculaMapa()` para repintar las zonas tras borrar el punto. **Debes localizar dónde hacer esta tarea.**

. ----------

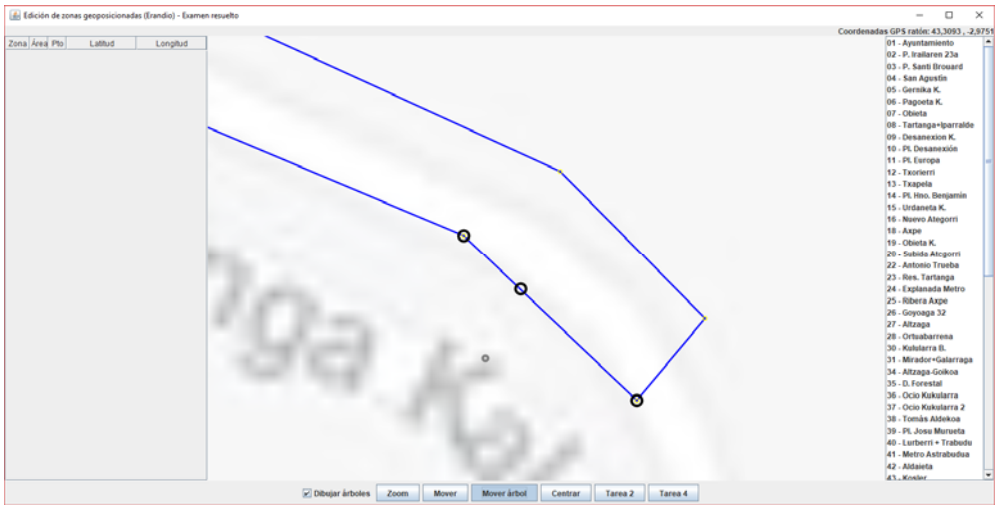
Tarea 4 [2,5]. **Bases de datos** (*Tareas.java*). Al pulsar el botón (tarea 4) queremos que se actualice una tabla que debe quedarse siempre solo con los **puntos de zonas que se están viendo en pantalla en ese momento.** Debes crear también el código de base de datos para ello (utilizando o no el existente). La tabla que debes crear se puede llamar **punto** y debe tener las columnas siguientes: zona string, latitud double, longitud double.
Te será útil convertir las coordenadas de pantalla (método `getPanelDibujo`) a latitud/longitud. Tienes métodos para convertir coordenadas ya programados en las líneas 557 en adelante de *EdicionZonasGPS*.

. ----------

Tarea 5 [1]. **Eventos** (*EdicionZonasGPS.java*). Observa que al mover el ratón aparece en un jlabel superior las coordenadas GPS por las que pasa. Añade código en el lugar indicado para que si se pulsa la tecla Control además de las coordenadas aparezca en ese jlabel la zona en la que se encuentra el ratón (o nada si sus coordenadas no corresponden a ninguna zona).

. ----------

Tarea 6 [2,5]. **Recursividad** (*EdicionZonasGPS.java*). Con el botón "Mover árbol" se puede mover un árbol en el mapa haciendo drag con el ratón sobre él. Queremos que tras hacerlo se pinte en el mapa el punto del borde de zona más cercano, intermedio a los puntos del segmento. Para ello tienes que programar dos partes:
[1 punto] Parte no recursiva: buscar los dos puntos más cercanos. Puedes hacer la simplificación de buscar los puntos más cercanos de todo el mapa. (Si no sabes cómo resolver esta parte puedes intentar solo la segunda)
[1,5 puntos] Parte recursiva: partiendo del árbol y de los dos vértices más cercanos, determinar recursivamente con 10 llamadas por aproximaciones sucesivas el punto más cercano del segmento entre esos dos puntos y dibujarlo. Puedes ver un ejemplo en esta figura:



(No te preocupes si la apariencia no es exacta: la longitud y la latitud no tienen la misma progresión, así que calcular la distancia utilizándolas sin conversión no es la mejor manera, pero vale para esta tarea)