EJEMPLO DE PARCIAL 2 RESUELTO

NOTAS PARA LA REALIZACIÓN DEL EJERCICIO:

- Al **inicio del contenido de cada fichero** realizado deberá aparecer un comentario con tus **apellidos y nombre, titulación y grupo.**
- Los diferentes apartados tienen una determinada puntuación. Si un apartado no lo sabes hacer, *no debes pararte en él indefinidamente*. Puedes abordar otros.
- **Está permitido:**
  - Consultar los apuntes (CV), y la guía rápida de la API (CV).
  - Añadir métodos privados a las clases.
- **No está permitido**:
  - Intercambiar documentación con otros compañeros.
  - Recibir ayuda de otras personas. Se debe realizar personal e individualmente la solución del ejercicio propuesto.
  - Añadir métodos no privados a las clases.
  - Añadir variables o constantes a las clases.
  - Modificar el código suministrado.
- Una vez terminado el ejercicio, debéis subir (a la tarea creada en el campus virtual para ello) un fichero comprimido de la carpeta **src** que hayáis realizado y usáis vuestros apellidos y nombre para su denominación **(Apellido1Apellido2Nombre.rar** o **.zip)**.
- La evaluación tendrá en cuenta la claridad de los algoritmos, del código y la correcta elección de las estructuras de datos, así como los criterios de diseño que favorezcan la reutilización.
- Para la corrección del ejercicio se utilizarán **programas de detección de copias/plagios**.
- Con posterioridad a la realización del ejercicio, el profesor podrá convocar a determinado/as alumno/as para realizar **entrevistas personales** con objeto de comprobar la autoría de las soluciones entregadas.

Proyecto prBus

Se va a crear una aplicación para mantener líneas de autobuses. Para ello se definirán las clases `Parada` y `LineaBus` en el paquete denominado `bus`, con las características indicadas en los siguientes ejercicios. Se proporciona una primera versión de la clase Parada, correspondiente a la solución de la prueba realizada el pasado 30 de marzo. Asimismo, el proyecto incluirá la interfaz `CriterioParadas` y la clase `AlNorteParadaInicial`.

(0,5 puntos) Defínase la clase `BusException` que represente excepciones no comprobadas para tratar las situaciones excepcionales en el proyecto.

(2,25 puntos) Se debe modificar la clase `Parada` para incorporar la siguiente funcionalidad:

a) La latitud debe siempre estar entre -90 y +90 grados, y la longitud entre -180 y +180 grados. Modifíquese adecuadamente el constructor de la clase para garantizarlo y que, en caso contrario, se lance la correspondiente excepción.

b) Dos paradas se considerarán iguales cuando coincidan sus nombres (sin tener en cuenta mayúsculas o minúsculas), sus latitudes y sus longitudes.

3) **(0,75 puntos)** Debe definirse una aplicación (clase distinguida) `PruebaParada`, en el paquete por omisión, que pruebe la clase anterior, creando dos objetos de la clase Parada, con los datos siguientes:

```
Paseo del Parque, 36.71884, -4.41910
paseo del parque, 36.71884, -4.41910
```

Debe imprimirse en pantalla un mensaje indicando si son paradas iguales o no.

A continuación, debe crearse un tercer objeto de la clase Parada con los datos:

```
Louis Pasteur, 36.71654, -184.47508
```

Y debe capturarse la excepción (por ser la longitud menor de -180), imprimiendo un mensaje sobre la variable `System.err`.

4) **(4 puntos)** La clase `LineaBus` deberá incluir información sobre el nombre de una línea (`String`), y una colección de paradas en su recorrido recogidas en una lista (`ArrayList<Parada>`). En este caso, la clase incluirá:
   a) Un constructor con el nombre de la línea como argumento, y que considere un recorrido inicial vacío, sin paradas.
   b) Un método para añadir una parada al final de la lista:
      `void agregar(Parada parada)`
   c) Un método
      `public int agregarParadas(String[] datosParadas)`
      que agrege las paradas cuya información se proporciona en el array que se pasa como argumento, donde cada elemento del array se supondrá con el formato:
      `nombre de la parada@latitud,longitud`
      El método devolverá como resultado el número de estos elementos del array que no tengan el formato correcto.
   d) Un método
      `public ArrayList<Parada> paradasAlejadas(double dist)`
      que devuelva una lista con todas las paradas de la línea que se encuentren, de la primera parada, a una distancia mayor o igual que `dist`.
   e) La representación textual de una línea de autobús vendrá dada por el nombre de la línea (todo en mayúsculas), seguido por la información de las paradas, encerradas entre llaves y separadas por " -> ". Utilícese la clase `StringJoiner` o `StringBuilder`, según sea más conveniente.

5) **(2,5 puntos)** Con objeto de no tener que definir en la clase `LineaBus` distintos métodos para filtrar las paradas atendiendo a diversos criterios, como el descrito en el apartado (d) del ejercicio anterior, se deben definir la siguiente interfaz, solo una de las clases que la implementan, apartado (b) o (c), y el método `filtrarParadas` en la clase `LineaBus`:
   a) La interfaz `CriterioParadas` incluirá un solo método:
      `ArrayList<Parada> seleccionar(ArrayList<Parada> paradas)`
      con la intención de devolver una lista de paradas que constituya una selección (siguiendo determinado criterio) de las paradas que se pasan como argumento. Los diferentes criterios serán definidos por clases (como las de los apartados siguientes) que implementen la interfaz.
   b) La clase `DistanciaMinima` implementará la interfaz anterior, de forma que incluirá:

i. Una variable de instancia (`double`) que almacenará una distancia de referencia.
ii. Un constructor que permita inicializar esa distancia.
iii. La implementación del método `seleccionar`, que filtre las paradas que se pasan como argumento, devolviendo una lista con las que distan de la parada inicial (la primera en la lista) más que la distancia de referencia.

c) La clase `AlNorte` implementará también la interfaz del apartado (a) de forma que el método `seleccionar` devolverá las paradas que estén al norte de la parada inicial (la primera en la lista de paradas). Una parada estará al norte de otra cuando su latitud sea mayor.

d) Incluir en la clase `LineaBus` el método:

```
ArrayList<Parada> filtrarParadas(CriterioParadas filtro)
```

que devuelva la lista de paradas seleccionadas según el filtro que se pase como argumento.

Se proporciona una clase de pruebas para incluir en el paquete por omisión: `PruebasAdicionales`, para ilustrar el uso de las clases propuestas.