FUNDAMENTOS DE PROGRAMACIÓN. Curso 2020/21 **SEGUNDA CONVOCATORIA. 15 de septiembre de 2021**	Aula: Asiento: ☐B ☐V nº__	Versión 1.0.4

APELLIDOS: _____ NOMBRE: _____

DNI: _____ TITULACIÓN: ☐ IC ☐ IS ☐ TI ☐ DG GRUPO: ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5

EJERCICIO 1 – TIPO BASE

Se desea implementar parte de un sistema relacionado con el registro de la actividad deportiva de diferentes usuarios. Para ello se define el tipo **RegistroActividad** con la siguiente descripción:

Propiedades:

- **usuario**, de tipo String, consultable. Nombre del usuario que realiza la actividad.
- **tipo**, de tipo TipoActividad, consultable y modificable. Tipo de actividad realizada. Puede tomar los valores: CAMINAR, RUNNING, NATACION, BICICLETA_ESTATICA, CICLISMO, PILATES, REMO.
- **registroRuta**, de tipo Boolean, consultable. Si es true indica que la actividad tiene registrados los puntos de la ruta seguida geolocalizados con GPS.
- **horas**, de tipo List<LocalDateTime>, consultable. Si la actividad tiene registrada la ruta, aquí se almacenan las fechas y horas en las que el usuario pasó por los distintos puntos de la ruta. Si la actividad no tiene registro de ruta, la lista solo tendrá dos elementos:la fecha y hora de inicio de la actividad (el primer elemento de la lista) y la fecha y hora de fin de la actividad (el último elemento de la lista).
- **coordenadas**, de tipo List<Coordenada>, consultable. Si la actividad tiene registrada la ruta, contiene las coordenadas (latitud y longitud) de los puntos de la ruta. Si la actividad no tiene registrada la ruta, estará vacía.
- **alturas**, de tipo List<Double>, consultable. Si la actividad tiene registrada la ruta, contiene las altitudes de los puntos de la ruta. Si la actividad no tiene registrada la ruta, estará vacía.
- **alturaMaxima**, de tipo Double, consultable. Es la altura máxima a la que ha subido el usuario en la actividad. Si la actividad no tiene registrada la ruta, será null.
- **distancia**, de tipo Double, consultable. Se calcula como la suma de las distancias en kilómetros que hay entre cada dos puntos consecutivos de la ruta. Si la actividad no tiene registrada la ruta, será 0.0.
- **duracion**, de tipo Duration, consultable. Duración de la actividad. Se calcula en base a la fecha y hora de comienzo de la actividad y la fecha y hora de finalización de la misma.
- **inicio**, de tipo LocalDateTime, consultable. La fecha y hora de inicio de la actividad.
- **fin**, de tipo LocalDateTime, consultable. La fecha y hora de finalización de la actividad.
- **fecha**, de tipo LocalDate, consultable. La fecha de inicio de la actividad.

Otras operaciones:

- **getCoordenadaPuntoRutaMasCercanoA**(Coordenada c, Double umbral), de tipo Coordenada. Dados una coordenada y un umbral, devuelve la coordenada más cercana a la coordenada c de todas aquellas que estén situadas a una distancia de la coordenada c inferior o igual al umbral. Si no hay ninguna coordenada cercana, se devuelve null.

Criterio de igualdad: dos objetos de tipo RegistroActividad son iguales si tienen el mismo usuario, el mismo tipo de actividad y la misma fecha y hora de comienzo.

Representación como cadena: el usuario, seguido de dos puntos, la fecha y hora de comienzo, seguida de un guion y el tipo de actividad. Ejemplo: “janedoe: 15/09/2021 08:30 - CAMINAR”

Constructores:

- C1: un constructor que toma como parámetros un nombre de usuario, un tipo de actividad, una fecha y hora de inicio, y una fecha y hora de fin. Crea un registro de actividad **sin registro de ruta**, cuyo usuario, tipo de actividad, fecha y hora de inicio y fecha y hora de fin son los dados como parámetros.
- C2: un constructor que toma como parámetros un nombre de usuario, un tipo de actividad, una lista de fechas y horas, una lista de coordenadas y una lista de alturas. Crea un registro de actividad **con registro de ruta**, cuyo usuario, tipo de actividad, fechas y horas, coordenadas y alturas de los puntos de la ruta son los dados como parámetros. (Puede usar si lo estima conveniente los métodos de la clase de utilidad Ckeckers)

Restricciones:

- R1: Las listas horas, coordenadas y alturas deben tener el mismo número de elementos, que debe ser superior o igual a 2, si la actividad tiene registro de ruta.
- R2: La lista de horas debe tener exactamente dos elementos, y las listas de coordenadas y alturas deben estar vacías, si la actividad no tiene registro de ruta.
- R3: Dada una fecha-hora cualquiera de la lista horas situada en el índice i, se debe cumplir que esa fecha-hora debe ser anterior a la fecha-hora de la posición i+1 y posterior a la fecha-hora de la posición i-1.
- R4: Las actividades de tipo PILATES y BICICLETA_ESTATICA no pueden tener registro de ruta.

Suponga definido el tipo Coordenada, con dos propiedades: latitud, de tipo Double, consultable y modificable; y longitud, de tipo Double, consultable y modificable. Y con la operación Double getDistancia(Coordenada cs), que devuelve la distancia (en kilómetros) entre la coordenada con la que se invoca al método y la coordenada cs.

Se pide implementar lo siguiente **SIN HACER USO** de Streams:

a) Cabecera y atributos de la clase y constructor C2. 1,5 puntos
b) Método getCoordenadaPuntoRutaMasCercanoA. 1 punto

EJERCICIO 2 – CLASE CONTENEDORA

El tipo contenedor RegistrosUsuarios tiene la siguiente descripción:

Propiedades:

- **registros**, de tipo List<RegistroActividad>. Consultable. Lista con los registros de actividad de diferentes usuarios. Un usuario puede tener registrada más de una actividad.

Constructores:

- Un constructor que tiene como parámetro un Stream<RegistroActividad> y crea un objeto de tipo RegistrosUsuarios a partir del Stream dado (se supone implementado).

Implemente las siguientes operaciones de este **tipo usando exclusivamente métodos de Stream**:

a) **getUsuariosRutasCercanas**: Dados una coordenada c, de tipo Coordenada, y un umbral u, de tipo Double, devuelve una lista con los nombres de usuarios distintos que tienen una actividad en la que han pasado por algún punto situado a una distancia menor o igual a u de la coordenada c. La lista estará ordenada alfabéticamente de manera ascendente por nombre de usuario. 1 punto.
b) **getMinutosActividadPorSemana**: Dado un usuario, devuelve un SortedMap<Integer, Long>, que asocia el total de minutos de actividad agrupado por semana del año para el usuario dado como parámetro y ordenado por orden natural de los números de la semana. Por ejemplo, si el resultado es {1:180, 2:200} significa que el usuario dado como parámetro tiene registrados 180 minutos de actividad en la primera semana del año y 200 minutos en la segunda semana del año. Para resolver este ejercicio dispone de un método auxiliar en la clase de utilidad UtilesFechas, que, dada una fecha, devuelve el número de la semana del año a la que corresponde esa fecha. Tenga en cuenta que el cálculo de la semana a la que se imputa una actividad se hace en base a la fecha de comienzo de esa actividad. El método auxiliar tiene la siguiente signatura: *static Integer getWeekOfYear(LocalDate fecha)*. 1,5 puntos
c) **getTipoActividadMasPracticada**: Devuelve el tipo de actividad más practicada de todas las actividades registradas. La actividad más practicada es aquella que es practicada por un mayor número de usuarios distintos, independientemente del número de actividades de ese tipo que registren los usuarios. 2,5 puntos.
d) **esUsuarioSaludableOMS**: Dado un usuario, devuelve true si ese usuario es saludable atendiendo a los criterios de la Organización Mundial de la Salud, que son los siguientes:
    - El usuario ha debido registrar actividad en al menos 48 semanas del año.
    - En cada semana con actividad ha debido realizar al menos 5 actividades.
    - El número de minutos de actividad por semana debe ser mayor o igual que 180.

    Tenga en cuenta que el cálculo de la semana a la que se imputa una actividad se hace en base a la fecha de comienzo de esa actividad. 2,5 puntos