Ejercicio 1. Apartado a)

```
public class RegistroActividad {

	private String usuario;
	private TipoActividad tipo;
	private Boolean registroRuta;
	private List<LocalDateTime> horas;
	private List<Coordenada> coordenadas;
	private List<Double> alturas;

	private static final String ERR_PUNTOS_RUTA =
	  "Las listas de horas, coordenadas y alturas deben tener el mismo número de elementos";
	private static final String ERR_HORAS_PUNTOS =
	  "Un punto cualquiera de la ruta debe tener una hora anterior al siguiente punto de la
	  ruta";
	private static final String ERR_NUM_PUNTOS_RUTA =
	  "Una ruta debe tener al menos dos puntos";
	private static final String ERR_ACTIVIDAD_SIN_RUTA =
	  "Las actividades PILATES y BICICLETA ESTÁTICA no pueden tener ruta";

	public RegistroActividad (String usuario, TipoActividad tipo,
			List<LocalDateTime> horas, List<Coordenada> coordenadas,
			List<Double> alturas) {
		Checkers.check (ERR_PUNTOS_RUTA, horas.size() == alturas.size()
			&& horas.size() == coordenadas.size());
		Checkers.check (ERR_NUM_PUNTOS_RUTA, horas.size() >= 2);
		Checkers.check(ERR_HORAS_PUNTOS, sonHorasOK(horas));

		this.usuario = usuario;
		this.registroRuta = true;
		setTipo(tipo);
		this.horas = new ArrayList<LocalDateTime>(horas);
		this.coordenadas = new ArrayList<Coordenada>(coordenadas);
		this.alturas = new ArrayList<Double>(alturas);
	}


	private Boolean sonHorasOK(List<LocalDateTime> horas) {
		Boolean res = true;
		for (int i = 0; i < horas.size() - 2; i++) {
			if (!horas.get(i).isBefore(horas.get(i+1))) {
				res = false;
				break;
			}
		}

		return res;
	}

	public void setTipo(TipoActividad tipo) {
		Checkers.check(ERR_ACTIVIDAD_SIN_RUTA, esTipoOK(tipo));
		this.tipo = tipo;
	}

	private Boolean esTipoOK(TipoActividad tipo) {
		return this.registroRuta.equals(true) && !tipo.equals(TipoActividad.PILATES)
			&& !tipo.equals(TipoActividad.BICICLETA_ESTATICA);
	}
```

Ejercicio 1. Apartado b)

```
public Coordenada getCoordenadaPuntoRutaMasCercanoA (Coordenada c, Double umbral) {
	Coordenada res = null;
	Double distRes = null;

	for (Coordenada coord: this.coordenadas) {
		Double d = coord.getDistancia(c);
		if (d <= umbral) {
			if (distRes == null || distRes > d) {
				res = coord;
				distRes = d;
			}
		}
	}
	return res;
}
```

Ejercicio 2. Apartado a)

```
public List<String> getUsuariosRutasCercanas(Coordenada c, Double umbral) {

	Comparator<RegistroActividad> comp =
		Comparator.comparing (RegistroActividad::getUsuario);

	return registros.stream()
		.filter(reg -> reg.getCoordenadaPuntoRutaMasCercanoA(c, umbral) != null)
		.sorted(comp)
		.map(RegistroActividad::getUsuario)
		.distinct()
		.collect(Collectors.toList());
}
```

Ejercicio 2. Apartado b)

```
public SortedMap<Integer, Long> getMinutosActividadPorSemana(String usuario) {
	return registros.stream()
		.filter(reg -> reg.getUsuario().equals(usuario))
		.collect(Collectors.groupingBy(
			reg -> UtilesFechas.getWeekOfYear(reg.getFecha()),
			TreeMap::new,
			Collectors.summingLong(reg -> reg.getDuracion().toMinutes())));
}
```

Ejercicio 2. Apartado c)

```
public TipoActividad getTipoActividadMasPracticada() {
	Map<TipoActividad, Set<String>> m = registros.stream()
		.collect(Collectors.groupingBy(RegistroActividad::getTipo,
			Collectors.mapping(RegistroActividad::getUsuario,
				Collectors.toSet())));

	Comparator<Entry<TipoActividad, Set<String>>> c =
		Comparator.comparing(e -> e.getValue().size());

	return m.entrySet().stream()
		.max(c)
		.get()
		.getKey();
}
```

Ejercicio 2. Apartado d)

```
public Boolean esUsuarioSaludableOMS(String usuario) {

	SortedMap<Integer, Long> actividad = getMinutosActividadPorSemana(usuario);
	SortedMap<Integer, Long> numActividades = getNumActividadesPorSemana(usuario);

	Predicate<Entry<Integer, Long>> pred = entry -> entry.getValue() >= 150L &&
		numActividades.get(entry.getKey()) >= 5L;

	return actividad.size() >= 48 && actividad.entrySet().stream()
		.allMatch(pred);
}

private SortedMap<Integer, Long> getNumActividadesPorSemana(String usuario) {

	return registros.stream()
		.filter(reg -> reg.getUsuario().equals(usuario))
		.collect(Collectors.groupingBy(
			reg -> UtilesFechas.getWeekOfYear(reg.getFecha()),
			TreeMap::new,
			Collectors.counting()));
}
```