



### EJERCICIO 1 (1 punto)

Restricciones (0,25 puntos)

```
Checkers.check("La escala Richter debe estar comprendida entre 1.5 y 12",  
    1.5 <= richter && Richter <= 12);  
Checkers.check("La fecha debe ser anterior o igual al día de hoy",  
    fecha.compareTo(LocalDate.now()) <= 0);
```

Orden natural (0,25 puntos)

```
public int compareTo(Registro r) {  
    int res = this.getFecha().compareTo(r.getFecha());  
    if (res == 0) {  
        res = this.getCoordenadas().compareTo(r.getCoordenadas());  
    }  
    return res;  
}
```

Método getDistancia (0,5 puntos)

```
public Double getDistancia(Coordenadas c) {  
    return getCoordenadas().getDistanciaHarvesine(c);  
}
```

### EJERCICIO 2 (8 puntos)

Tipo contenedor, operaciones básicas (0,5 puntos)

```
public class Registros {  
  
    private List<Registro> registros;  
  
    public Registros() {  
        registros = new ArrayList<Registro>();  
    }  
  
    public Registros(Stream<Registro> sreg) {  
        this.registros = sreg.collect(Collectors.toList());  
    }  
  
    public Integer getNumRegistros() {  
        return registros.size();  
    }  
  
    public SortedSet<Registro> getRegistros() {  
        return new TreeSet<>(registros);  
    }  
}
```

a) Método getCiudadesIntensidadMayorEnAnyo (1 punto)

```
public Set<String> getCiudadesIntensidadMayorEnAnyo(
    Double umbralRichter, Integer anyo) {
    return registros.stream()
        .filter(r -> r.getRichter() > umbralRichter
            && r.getFecha().getYear() == anyo)
        .map(Registro::getCiudad)
        .collect(Collectors.toSet());
}
```

b) Método calcularPromedioRichterConjuntoCiudades (1 punto)

```
public Double calcularPromedioRichterConjuntoCiudades(
    Set<String> ciudades) {
    return registros.stream()
        .filter(r -> ciudades.contains(r.getCiudad()))
        .mapToDouble(Registro::getRichter)
        .average()
        .getAsDouble();
}
```

Alternativa:

```
public Double calcularPromedioRichterConjuntoCiudades(
    Set<String> ciudades) {
    return registros.stream()
        .filter(r -> ciudades.contains(r.getCiudad()))
        .collect(Collectors.averagingDouble(Registro::getRichter));
}
```

c) Método getCoordenadasPorCiudad (1 punto)

```
public Map<String, List<Coordenadas>> getCoordenadasPorCiudad() {
    return registros.stream()
        .collect(Collectors.groupingBy(Registro::getCiudad,
            Collectors.mapping(Registro::getCoordenadas,
                Collectors.toList())));
}
```

d) Método getCoordenadasMediasPorCiudad (1,5 puntos)

```
public Map<String, Coordenadas> getCoordenadasMediasPorCiudad() {
    Map<String, List<Coordenadas>> mpaux = getCoordenadasPorCiudad();
    Map<String, Coordenadas> mp = new HashMap<>();

    for (String ciudad: mpaux.keySet()) {
        mp.put(ciudad, getCoordenadasMedias(mpaux.get(ciudad)));
    }

    return mp;
}
```

```

private Coordenadas getCoordenadasMedias(List<Coordenadas> lc) {
    Double latmedia = lc.stream()
        .collect(Collectors.averagingDouble(Coordenadas::getLatitud));
    Double lonmedia = lc.stream()
        .collect(Collectors.averagingDouble(Coordenadas::getLongitud));

    return new Coordenadas(latmedia,lonmedia);
}

```

Alternativa 1:

```

public Map<String, Coordenadas> getCoordenadasMediasPorCiudad() {

    Map<String, List<Coordenadas>> mpaux = getCoordenadasPorCiudad();
    Map<String,Coordenadas> mp = new HashMap<>();

    for (Map.Entry<String, List<Coordenadas>> par: mpaux.entrySet()) {
        mp.put(par.getKey(), getCoordenadasMedias(par.getValue()));
    }

    return mp;
}

```

Alternativa 2:

```

public Map<String, Coordenadas> getCoordenadasMediasPorCiudad() {
    Map<String, List<Coordenadas>> m = getCoordenadasPorCiudad();

    return m.entrySet().stream()
        .collect (Collectors.toMap(
            Map.Entry::getKey,
            entry -> getCoordenadasMedias(entry.getValue())));
}

```

e) Método getCiudadMasTerremotosNMayores (1,5 puntos)

```

public String getCiudadMasTerremotosNMayores(Integer n) {
    Map<String, Long> mp = getNumTerremotosPorCiudadNMayores(n);

    return mp.entrySet().stream()
        .max(Comparator.comparing(x -> x.getValue()))
        .get()
        .getKey();
}

private Map<String, Long> getNumTerremotosPorCiudadNMayores(Integer n) {
    return registros.stream()
        .sorted(Comparator.comparing(Registro::getRichter).reversed())
        .limit(n)
        .collect(Collectors.groupingBy(Registro::getCiudad,
            Collectors.counting()));
}

```

f) Método getMediaDiasEntreTerremotosCiudad (1,5 puntos)

```
public Double getMediaDiasEntreTerremotosCiudad(String ciudad) {
    Double res = 0.0;
    List<LocalDate> fechas = getFechasTerremotosOrdenadas(ciudad);
    Double suma = 0.;

    for (int i = 0; i < fechas.size() - 1 ; i++){
        long diff = fechas.get(i).until(fechas.get(i + 1),
            ChronoUnit.DAYS);
        suma = suma + diff;
    }

    if (fechas.size() - 1 > 0) {
        res = suma / (fechas.size() - 1);
    }
    return res;
}

private List<LocalDate> getFechasTerremotosOrdenadas(String ciudad) {
    return registros.stream()
        .filter(r -> r.getCiudad().equals(ciudad))
        .sorted(Comparator.comparing(r -> r.getFecha()))
        .map(r -> r.getFecha())
        .collect(Collectors.toList());
}
```

### EJERCICIO 3 (1 punto)

```
private static Registro parsearRegistro(String linea) {
    String[] trozos = linea.split(",");
    Checkers.check("Debe haber 5 atributos", trozos.length == 5);

    LocalDate fecha = LocalDate.parse(trozos[0].trim(),
        DateTimeFormatter.ofPattern("yyyy.M.d"));
    Double latitud = Double.valueOf(trozos[1].trim());
    Double longitud = Double.valueOf(trozos[2].trim());
    Coordenadas coordenadas = new Coordenadas(latitud, longitud);
    String ciudad = trozos[3].trim();
    Double richter = Double.valueOf(trozos[4].trim());

    return new Registro(fecha, coordenadas, ciudad, richter);
}
```