FUNDAMENTOS DE PROGRAMACIÓN. Curso 2018/19

SEGUNDO EXAMEN PRÁCTICO. 12 de junio de 2019. Soluciones

 # EJERCICIO 1 [1,5 puntos]

Comprobar restricciones en el constructor (0,5 puntos)

```
	public Registro(Integer anyo, String pais, Double puntos, Double
				gobierno, Double justicia, Double derechos) {

		Checkers.check("El año debe estar entre 2009 y 2016",
				anyo >= 2009 && anyo <= 2016);
		Checkers.check("Los puntos deben estar entre 0 y 10 o tomar el valor -1",
				(puntos >= 0 && puntos <= 10) || puntos == -1);
		Checkers.check("El gobierno debe estar entre 0 y 10 o tomar el valor -1",
				(gobierno >= 0 && gobierno <= 10) || gobierno == -1);
		Checkers.check("La justicia debe estar entre 0 y 10 o tomar el valor -1",
				(justicia >= 0 && justicia <= 10) || justicia == -1);
		Checkers.check("Los derechos deben estar entre 0 y 10 o tomar el valor -1",
				(derechos >= 0 && derechos <= 10) || derechos == -1);
		this.anyo = anyo;
		this.pais = pais;
		this.puntos = puntos;
		this.gobierno = gobierno;
		this.justicia = justicia;
		this.derechos = derechos;
	}
```

Método getNivel (0,5 puntos)

```
	public Nivel getNivel() {
		Nivel res;
		if (puntos >= gobierno && puntos >= justicia) {
			res = Nivel.ALTO;
		} else if (puntos <= gobierno && puntos <= justicia) {
			res = Nivel.BAJO;
		} else {
			res = Nivel.MEDIO;
		}
		return res;
	}
```

Método compareTo (0,5 puntos)

```
	public int compareTo(Registro r) {
		int res = anyo.compareTo(r.getAnyo());
		if (res == 0) {
			res = pais.compareTo(r.getPais());
		}
		return res;
	}
```

 # EJERCICIO 2 [0,5 puntos]

```
public class Registros {

	private SortedSet<Registro> registros;

	public Registros(Stream<Registro> s) {
		this.registros = s.collect(Collectors.toCollection(TreeSet::new));
```

 # EJERCICIO 3 [0,5 puntos]

```
	private static Registro parsearRegistro(String lineaCSV) {
		String[] sp = lineaCSV.split(";");
		Checkers.check("Cadena formato no válido", sp.length == 6);

		Integer anyo = Integer.valueOf(sp[0].trim());
		String pais = sp[1].trim();
		Double puntos = Double.valueOf(sp[2].trim());
		Double gobierno = Double.valueOf(sp[3].trim());
		Double justicia = Double.valueOf(sp[4].trim());
		Double derechos = Double.valueOf(sp[5].trim());

		return new Registro(anyo, pais, puntos, gobierno, justicia, derechos);
	}
```

 # EJERCICIO 4 [7,5 puntos]

Método getValorMedioPuntuacionConjuntoPaises (1 punto)

```
	public Double getValorMedioPuntuacionConjuntoPaises(Set<String> paises) {
		return registros.stream()
			.filter(r -> paises.contains(r.getPais()))
			.mapToDouble(Registro::getPuntos)
			.average()
			.getAsDouble();
	}
```

Método getPaisMayorPuntuacion (1 punto)

```
	public String getPaisMayorPuntuacion(Integer anyo) {
		return registros.stream()
			.filter(r -> r.getAnyo().equals(anyo))
			.max(Comparator.comparing(Registro::getPuntos))
			.get()
			.getPais();
	}
```

Método getMediaDerechosPorPais (1 punto)

```
	public Map<String, Double> getMediaDerechosPorPais() {
		return registros.stream()
			.filter(r -> !r.getDerechos().equals(-1.0))
			.collect(Collectors.groupingBy(
				Registro::getPais,
				Collectors.averagingDouble(Registro::getDerechos))
			);
	}
```

Método getPaisConMenorMediaDerechos (1 punto)

```
	public String getPaisConMenorMediaDerechos() {
		return getMediaDerechosPorPais()
			.entrySet().stream()
			.min(Comparator.comparing(e -> e.getValue()))
			.get()
			.getKey();
	}

	// Alternativa
	public String getPaisConMenorMediaDerechos() {
		Map<String, Double> mediaDerechosPorPais = getMediaDerechosPorPais();
		return mediaDerechosPorPais
			.keySet().stream()
			.min(Comparator.comparing(p -> mediaDerechosPorPais.get(p)))
			.get();
	}
```

Método getPaisMasDerechosPorAnyo (1,5 puntos)

```
	public Map<Integer, String> getPaisMasDerechosPorAnyo() {
		return registros.stream()
			.collect(Collectors.groupingBy(
				Registro::getAnyo,
				Collectors.collectingAndThen(
				   Collectors.maxBy(Comparator.comparing(Registro::getDerechos)),
				   o -> o.get().getPais()))
			);
	}
```

Método getListasNPaisesPorAnyo (2 puntos)

```java
	// Método principal (0,5 puntos)
	public Map<Integer, List<String>> getListaNPaisesPorAnyo(Integer n) {
		Map<Integer, List<Registro>> registrosPorAnyo = getRegistrosPorAnyo();

		Map<Integer, List<String>> res = new HashMap<>();
		for (Integer a: registrosPorAnyo.keySet()) {
			List<String> paises = getNPaisesOrdenados(registrosPorAnyo.get(a), n);
			res.put(a, paises);
		}
		return res;
	}

	// Método auxiliar 1 (0,5 puntos)
	private Map<Integer, List<Registro>> getRegistrosPorAnyo() {
		return registros.stream()
			.collect(Collectors.groupingBy(Registro::getAnyo));
	}

	// Método auxiliar 2 (1 punto)
	private List<String> getNPaisesOrdenados(List<Registro> l, Integer n) {
		return l.stream()
			.sorted(Comparator.comparing(Registro::getJusticia).reversed())
			.limit(n)
			.map(Registro::getPais)
			.collect(Collectors.toList());
	}

	// Alternativa para el método principal
	public Map<Integer, List<String>> getListaNPaisesPorAnyo(Integer n) {
		return getRegistrosPorAnyo()
			.entrySet()
			.stream()
			.collect(Collectors.toMap(
				x -> x.getKey(),
				x -> getNPaisesOrdenados(x.getValue(), n))
			);
	}
```