Ejercicio 1

```
public record Idioma(String nombre, Double porcentaje) {
	public Idioma {
		Checkers.check("El porcentaje no está entre 0 y 1",
			porcentaje >= 0.0 && porcentaje <= 1.0);
	}
	public int hashCode() {
		return Objects.hash(nombre);
	}
	public boolean equals(Object obj) {
		if (this == obj)
			return true;
		if (obj == null)
			return false;
		if (getClass() != obj.getClass())
			return false;
		Idioma other = (Idioma) obj;
		return Objects.equals(nombre, other.nombre);
	}
}
```

Ejercicio 2

```
public class Pais {
	private String codigoISO;
	private String nombre;
	private Long poblacion;
	private Double area;
	private String capital;
	private Continente continente;
	private String moneda;
	private List<Idioma> idiomas;
	private Set<String> vecinos;

	public Pais(String codigoISO, String nombre, Long poblacion, Double area,
			String capital, Continente continente, String moneda,
			Set<Idioma> idiomas, Set<String> vecinos) {
		Checkers.check("No hay capital o moneda para país con población",
			(poblacion > 0 && capital != null && moneda != null) || (poblacion==0));
		this.codigoISO = codigoISO;
		this.nombre = nombre;
		this.poblacion = poblacion;
		Checkers.check("El área debe ser mayor que cero", area >= 0.0);
		this.area = area;
		this.capital = capital;
		this.continente = continente;
		this.moneda = moneda;
		this.idiomas = new ArrayList<>(idiomas);
		this.vecinos = new HashSet<>(vecinos);
	}
```

```
public String getCodigoISO() {
	return codigoISO;
}

public String getNombre() {
	return nombre;
}

public Long getPoblacion() {
	return poblacion;
}

public Double getArea() {
	return area;
}

public String getCapital() {
	return capital;
}

public Continente getContinente() {
	return continente;
}

public String getMoneda() {
	return moneda;
}

public List<Idioma> getIdiomas() {
	return new ArrayList<>(idiomas);
}

public Set<String> getVecinos() {
	return new HashSet<>(vecinos);
}

public Boolean esAislado() {
	return vecinos.isEmpty();
}

public Double getDensidadPoblacion() {
	Double res = 0.0;
	if (area > 0) {
		res = poblacion / area;
	}
	return res;
}

public Boolean sonTodosVecinos(Set<Pais> paises) {
	Boolean res = true;
	for (Pais p: paises) {
		if (!getVecinos().contains(p.getCodigoISO())) {
			res = false;
			break;
		}
```

```
		}
		return res;
	}

	// Solución alternativa
	public Boolean sonTodosVecinos(Set<Pais> paises) {
		return paises.stream()
			.allMatch(p-> getVecinos().contains(p.getCodigoISO()));
	}

	public int hashCode() {
		return Objects.hash(codigoISO);
	}

	public boolean equals(Object obj) {
		if (this == obj)
			return true;
		if (obj == null)
			return false;
		if (getClass() != obj.getClass())
			return false;
		Pais other = (Pais) obj;
		return Objects.equals(codigoISO, other.codigoISO);
	}

	public String toString() {
		return "Pais [codigoISO=" + codigoISO + ", nombre =" + nombre + ",
			poblacion =" + poblacion + ", area=" + area + ",
			capital=" + capital + ", continente=" + continente + ",
			moneda=" + moneda + ", idiomas=" + idiomas + ",
			vecinos=" + vecinos + "]";
	}
}
```

Ejercicio 3

```
private static final String SEPARADOR_PRINCIPAL = ";";
private static final String SEPARADOR_SECUNDARIO = ",";
private static final String SEPARADOR_TERCIARIO = ":";

public static Pais parseaPais(String cadena) {
	String[] info = cadena.split(SEPARADOR_PRINCIPAL);
	Checkers.check("Formato no válido", info.length == 9);
	Long pob = Long.valueOf(info[2].trim());
	Double area = Double.valueOf(info[3].trim());
	String capital = parseaCadenaNula(info[4]);
	Continente c = parseaContinente(info[5].trim());
	String moneda = parseaCadenaNula(info[6]);
	List<Idioma> idiomas = parseaIdiomas(info[7].trim());
	Set<String> vecinos = parseaVecinos(info[8]);
	return new Pais(info[0].trim(), info[1].trim(), pob, area, capital, c,
		moneda, idiomas, vecinos);
}
```

```
private static Set<String> parseaVecinos(String cadena) {
	Set<String> res = new HashSet<>();
	if (!cadena.trim().isEmpty()) {
		String[] info = cadena.split(SEPARADOR_SECUNDARIO);
		for (String s: info) {
			res.add(s.trim());
		}
	}
	return res;
}

private static String parseaCadenaNula(String cadena) {
	String res = null;
	if (!cadena.trim().isEmpty()) {
		res = cadena.trim();
	}
	return res;
}

private static List<Idioma> parseaIdiomas(String cadena) {
	List<Idioma> res = new ArrayList<>();
	if (!cadena.trim().isEmpty()) {
		String[] info = cadena.split(SEPARADOR_SECUNDARIO);
		for (String s: info) {
			res.add(parseaIdioma(s.trim()));
		}
	}
	return res;
}

private static Idioma parseaIdioma(String cadena) {
	String[] info = cadena.split(SEPARADOR_TERCIARIO);
	Checkers.check("Formato idioma no válido <" + cadena + ">", info.length == 2);
	Double porcentaje = Double.parseDouble(info[1].trim());
	return new Idioma(info[0].trim(), porcentaje);
}

private static Continente parseaContinente(String s) {
	switch(s) {
		case "AF": return Continente.AFRICA;
		case "AN": return Continente.ANTARCTICA;
		case "AS": return Continente.ASIA;
		case "EU": return Continente.EUROPE;
		case "NA": return Continente.NORTH_AMERICA;
		case "OC": return Continente.OCEANIA;
		case "SA": return Continente.SOUTH_AMERICA;
		default: return null;
	}
}
```

Ejercicio 4.1

```
public Double porcentajeAreaDeContinente(Continente continente) {
	Double areaMundo = paises.stream()
		.mapToDouble(Pais::getArea)
		.sum();
	Double areaContinente = paises.stream()
		.filter(pais -> pais.getContinente().equals(continente))
		.mapToDouble(Pais::getArea)
		.sum();
	return areaContinente / areaMundo;
}

// Solución alternativa
public Double porcentajeAreaDeContinente(Continente continente) {
	Double areaMundo = paises.stream()
		.collect(Collectors.summingDouble(Pais::getArea));
	Double areaContinente = paises.stream()
		.filter(pais -> pais.getContinente().equals(continente))
		.collect(Collectors.summingDouble(Pais::getArea));
	return areaContinente / areaMundo;
}
```

Ejercicio 4.2

```
public List<String> nombresDeVecinos(String pais) {
	Map<String, String> paisesPorISO = paises.stream()
		.collect(Collectors.toMap(Pais::getCodigoISO, Pais::getNombre));
	Pais paisBuscado = paises.stream()
		.filter(p -> p.getNombre().equals(pais))
		.findFirst()
		.get();
	return paisBuscado.getVecinos().stream()
		.map(isoVecino -> paisesPorISO.get(isoVecino))
		.sorted()
		.toList();
}

// Solución alternativa
public List<String> nombresDeVecinos(String pais) {
	Pais paisBuscado = paises.stream()
		.filter(p -> p.getNombre().equals(pais))
		.findFirst()
		.get();
	return paisBuscado.getVecinos().stream()
		.map(isoVecino -> buscarPaisPorISO(isoVecino))
		.sorted()
		.toList();
}
private String buscarPaisPorISO(String iso) {
	return paises.stream()
		.filter(p -> p.getCodigoISO().equals(iso))
		.findFirst()
		.map(Pais::getNombre)
```

```
            .get();
}

// Solución alternativa
public List<String> nombresDeVecinos(String pais) {
      Set<String> res = paises.stream()
            .filter(p -> pais.equals(p.getNombre()))
            .flatMap(p -> p.getVecinos().stream())
            .collect(Collectors.toSet());
      return paises.stream()
            .filter(p -> res.contains(p.getCodigoISO()))
            .map(p -> p.getNombre())
            .sorted()
            .toList();
   }
```

Ejercicio 4.3

```
public SortedMap<String, List<String>> paisesPorIdiomas() {
      SortedMap<String, List<String>> res = new TreeMap<>();
      for (Pais pais: paises) {
            for (Idioma idioma: pais.getIdiomas()) {
                  String clave = idioma.nombre();
                  if (res.containsKey(clave)) {
                        res.get(clave).add(pais.getNombre());
                  } else {
                        List<String> paises = new ArrayList<String>();
                        paises.add(pais.getNombre());
                        res.put(clave, paises);
                  }
            }
      }
      for (List<String> paises: res.values()) {
            Collections.sort(paises);
      }
      return res;
}
```

Ejercicio 4.4

```
public SortedMap<String, String> paisMayorPoblacionPorMoneda() {
      Comparator<Pais> c = Comparator.comparing(Pais::getPoblacion)
            .thenComparing(Pais::getDensidadPoblacion);
      return paises.stream()
            .collect(Collectors.groupingBy(
                  Pais::getMoneda,
                  TreeMap::new,
                  Collectors.collectingAndThen(
                        Collectors.maxBy(c),
                        optPais -> optPais.get().getNombre())));
}
```

Ejercicio 4.5

```
public List<String> vecinosMasFrecuentes(Integer n) {
	Map<String, Integer> numVecinosPorPais = paises.stream()
		.collect(Collectors.toMap(
			Pais::getNombre,
			pais -> pais.getVecinos().size()));

	Comparator<Map.Entry<String, Integer>> c = Map.Entry.comparingByValue();
	return numVecinosPorPais.entrySet().stream()
		.sorted(c.reversed())
		.limit(n)
		.map(Map.Entry::getKey)
		.sorted()
		.toList();
	}
}

// Solución alternativa
public List<String> vecinosMasFrecuentes(Integer n) {
	Comparator<Pais> cmp = Comparator.comparing((Pais p) -> p.getVecinos().size())
		.reversed();
	return paises.stream()
		.sorted(cmp)
		.limit(n)
		.map(p -> p.getNombre())
		.sorted()
		.collect(Collectors.toList());

}
```