Ejercicio 1. Apartado a)

```
public class RegistroDeuda implements Comparable<RegistroDeuda> {

	private static final LocalDate ANYO_REF = LocalDate.of(1945, 1, 1);
	private static final String DELIM = ",";
	private String nombrePais;
	private LocalDate fecha;
	private Integer deudaTotal;
	private Double deudaPorcentajePIB;
	private Integer deudaPerCapita;

	public RegistroDeuda(String string) {
		String[] splits = string.split(DELIM);
		Checkers.check("", splits.length == 5);

		String nombre = splits[0].trim();
		Integer anyo = Integer.parseInt(splits[1].trim());
		LocalDate fecha = LocalDate.of(anyo, 1, 1);
		Integer total = Integer.parseInt(splits[2].trim());
		Double porcentaje = Double.parseDouble(splits[3].trim());
		Integer perCap = Integer.parseInt(splits[4].trim());

		Checkers.check("", fecha.compareTo(ANYO_REF) >= 0);
		Checkers.check("", total >= 0);
		Checkers.check("", porcentaje >= 0);
		Checkers.check("", perCap >= 0);

		this.nombrePais = nombre;
		this.fecha = fecha;
		this.deudaTotal = total;
		this.deudaPorcentajePIB = porcentaje;
		this.deudaPerCapita = perCap;
	}
```

Ejercicio 1. Apartado b)

```
	public Integer getPoblacionAproximada() {
		Double aux = (getDeudaTotal().doubleValue() / getDeudaPerCapita());
		return (int) Math.ceil(aux);
	}

	public Integer getPIBAproximado() {
		Double aux = getDeudaTotal() * getDeudaPorcentajePIB() / 100;
		return (int) Math.floor(aux);
	}

	public int compareTo(RegistroDeuda r) {
		int result = getNombrePais().compareTo(r.getNombrePais());
		if (result == 0) {
			result = getFecha().compareTo(r.getFecha());
		}
		return result;
	}
```

Ejercicio 2. Apartado a)

```
public SortedSet<String> getNombrePaisesEmpiezanCon(String prefix) {
	return registros.stream()
		.filter(x ->
			x.getNombrePais().toLowerCase().startsWith(prefix.toLowerCase()))
		.map(RegistroDeuda::getNombrePais)
		.collect(Collectors.toCollection(TreeSet::new));
}
```

Ejercicio 2. Apartado b)

```
public String getPaisMasDeudaPerCapita(List<String> paises, Integer year) {
	return registros.stream()
		.filter(registro -> registro.getFecha().getYear() == year &&
			paises.contains(registro.getNombrePais()))
		.max(Comparator.comparing(RegistroDeuda::getDeudaPerCapita))
		.map(RegistroDeuda::getNombrePais)
		.orElse(null);
}
```

Ejercicio 2. Apartado c)

```
public SortedMap<String, List<RegistroDeuda>> getRegistrosPorPaisEnOrdenCronologico() {
	return registros.stream()
		.collect(Collectors.groupingBy(
			RegistroDeuda::getNombrePais,
			TreeMap::new,
			Collectors.collectingAndThen(Collectors.toList(),
				lista -> lista.stream().sorted( Comparator.reverseOrder())
					.collect(Collectors.toList()))));
}
```

Solución alternativa

```
public SortedMap<String, List<RegistroDeuda>> getRegistrosPorPaisEnOrdenCronologico2() {
	Map<String, List<RegistroDeuda>> m = registros.stream()
		.collect(Collectors.groupingBy(RegistroDeuda::getNombrePais));

	return m.entrySet().stream()
		.collect(Collectors.toMap(entry -> entry.getKey(),
			entry-> ordenarLista(entry.getValue()),
				(lista1, lista2) -> lista1,
				TreeMap::new));
}

private List<RegistroDeuda> ordenarLista( List<RegistroDeuda> listaRegistros) {
	return listaRegistros.stream()
		.sorted(Comparator.reverseOrder())
		.collect(Collectors.toList());
}
```

Ejercicio 2. Apartado d)

```
public SortedMap<String, Double> getPorcentajeCambioDeudaTotalPorPaises() {
	SortedMap<String, List<RegistroDeuda>> m =
		getRegistrosPorPaisEnOrdenCronologico();
	return m.entrySet().stream()
		.filter(entry ->entry.getValue().size() > 1)
		.collect(Collectors.toMap(
			entry -> entry.getKey(),
			entry -> calculaDif(entry.getValue()),
			(entry1, entry2) -> entry1,
			TreeMap::new));
}
```

Solución alternativa

```
public SortedMap<String, Double> getPorcentajeCambioDeudaTotalPorPaises2() {
	SortedMap<String, List<RegistroDeuda>> m =
		getRegistrosPorPaisEnOrdenCronologico();
	return new TreeMap<String, Double>(
		m.keySet().stream()
			.filter(x -> m.get(x).size() > 1)
			.collect(Collectors.toMap((String x) -> x,
				(String x) -> calculaDif(m.get(x)))));
}

private Double calculaDif(List<RegistroDeuda> datos) {
	RegistroDeuda fin = datos.get(0);
	RegistroDeuda inicio = datos.get(datos.size() - 1);
	return 100.0 * (fin.getDeudaTotal() - inicio.getDeudaTotal()) /
		inicio.getDeudaTotal();
}
```