Ejercicio 1

```
public record CorteElectrico(String descripcion, LocalDateTime fechaInicio,
             LocalDateTime fechaRestablecimiento, String compañia, String region,
             Double perdida, Integer consumidores, List<String> etiquetas)
             implements Comparable<CorteElectrico> {

       static final Integer PERDIDA_CRITICA = 200;

       public CorteElectrico {
          Checkers.check(
              "La fecha de restablecimiento debe ser igual o posterior a la de inicio",
              !fechaRestablecimiento.isBefore(fechaInicio));
              // También fechaInicio.compareTo(fechaRestablecimiento) <= 0
          Checkers.check(
              "El número de consumidores afectados debe ser mayor o igual que 0, o null",
              consumidores == null || consumidores >= 0);
          Checkers.check("La lista de etiquetas debe contener al menos una etiqueta",
              etiquetas.size() > 0);
       }

       public Nivel severidad() {
              Nivel res = Nivel.BAJO;
              if (consumidores != null) {
                     if (consumidores >= 10000 && consumidores <= 100000) {
                            res = Nivel.MEDIO;
                     } else if (consumidores > 100000) {
                            res = Nivel.ALTO;
                     }
              }
              return res;
       }

       public Boolean esCritico() {
              Integer duracion = (int) fechaInicio.until(fechaRestablecimiento,
                     ChronoUnit.HOURS);
              Boolean res = false;
              if (perdida != null) {
                     res = perdida > PERDIDA_CRITICA || duracion > 10;
              }
              return res;
       }

       // También
       public Boolean esCritico() {
              return perdida != null &&
                     (perdida > 200 ||
                     Duration.between(fechaRestablecimiento, fechaInicio).toHours()>10);
       }
       public int compareTo(CorteElectrico c) {
              int res = fechaInicio.compareTo(c.fechaInicio());
              if (res == 0) {
                     res = region.compareTo(c.region());
              }
              return res;
       }
}
```

Ejercicio 2

```
public class InformeCortes {
	private String nombre;
	private LocalDate fecha;
	private List<CorteElectrico> cortes;

	public InformeCortes(String nombre, LocalDate fecha) {
		this.nombre = nombre;
		this.fecha = fecha;
		this.cortes = new ArrayList<>();
	}

	public InformeCortes(String nombre, LocalDate fecha, Stream<CorteElectrico> s) {
		this.nombre = nombre;
		this.fecha = fecha;
		this.cortes = s.collect(Collectors.toList());
	}

	public String getNombre() {
		return nombre;
	}

	public LocalDate getFecha() {
		return fecha;
	}

	public List<CorteElectrico> getCortes() {
		return new ArrayList<>(cortes);
	}
	public Integer getNumeroCortes() {
		return cortes.size();
	}
	public void incorporaCorte(CorteElectrico c) {
		if (!cortes.contains(c)) {
			cortes.add(c);
		}
	}
	public void incorporaCortes(List<CorteElectrico> cortes) {
		this.cortes.addAll(cortes);
	}
	public void eliminaCorte(CorteElectrico c) {
		cortes.remove(c);
	}
```

Ejercicio 3

```
	private static CorteElectrico parsearCorte(String lineaCSV) {
		String[] sp = lineaCSV.split(";");
		Checkers.check("Cadena con formato no válido", sp.length == 10);

		String descripcion = sp[0].trim();

		LocalDateTime fechaInicio = parseaFechaHora(sp[1], sp[2]);
		LocalDateTime fechaRestablecimiento = parseaFechaHora(sp[3], sp[4]);

		String compañia = sp[5].trim();
		String region = sp[6].trim();

		Double perdida = parseaDouble(sp[7]);
		Integer consumidores = parseaEntero(sp[8]);

		List<String> etiquetas = parseaEtiquetas(sp[9]);

		return new CorteElectrico(descripcion, fechaInicio,
				fechaRestablecimiento, compañia, region,
				perdida, consumidores, etiquetas);
	}

	private static LocalDateTime parseaFechaHora(String sf, String st) {
		LocalDateTime fechaHora = LocalDateTime.parse(sf.trim() + "-" + st.trim(),
				DateTimeFormatter.ofPattern("M/d/y-H:m"));
		return fechaHora;
	}

	private static Double parseaDouble (String d) {
		Double res = null;
		if (!d.trim().equals("Unknown")) {
			res = Double.valueOf(d.trim());
		}
		return res;
	}

	private static Integer parseaEntero (String e) {
		Integer res = null;
		if (!e.trim().equals("Unknown")) {
			res = Integer.valueOf(e.trim());
		}
		return res;
	}

	private static List<String> parseaEtiquetas(String s) {
		String[] array = s.trim().split(",");
		List<String> etiquetas = new ArrayList<>();
		for (String e: array) {
			etiquetas.add(e.trim());
		}
		return etiquetas;
	}
```

Ejercicio 4

Ejercicio 4.1

```
public Double mediaAfectadosEnRegiones(Nivel s, Set<String> regiones) {
	return cortes.stream()
		.filter(c -> c.consumidores() != null
				&& c.severidad().equals(s)
				&& regiones.contains(c.region()))
		.mapToInt(CorteElectrico::consumidores)
		.average()
		.getAsDouble();
}
```

Ejercicio 4.2

```
public List<String> compañiasCortesMasRecientes(String etiqueta, Integer n) {
	return cortes.stream()
		.filter(c -> c.etiquetas().contains(etiqueta))
		.sorted(Comparator.comparing(CorteElectrico::fechaInicio).reversed())
		.map(CorteElectrico::compañia)
		.distinct()
		.limit(n)
		.collect(Collectors.toList());
}
```

Ejercicio 4.3

```
// Con bucles
public SortedMap<String,SortedSet<String>> compañiasConCortesCriticosPorRegionB(){
	SortedMap<String, SortedSet<String>> res = new TreeMap<>();

	for (CorteElectrico c: cortes) {
		if (c.esCritico()) {
			String k = c.region();
			if (res.containsKey(k)) {
				res.get(k).add(c.compañia());
			} else {
				SortedSet<String> ss = new TreeSet<>();
				ss.add(c.compañia());
				res.put(k, ss);
			}
		}
	}
	return res;
}
// Con streams
public SortedMap<String,SortedSet<String>> compañiasConCortesCriticosPorRegionS(){
	return cortes.stream()
		.filter(c -> c.esCritico())
		.collect(Collectors.groupingBy(
				CorteElectrico::region,
				TreeMap::new,
				Collectors.mapping(
					CorteElectrico::compañia,
					Collectors.toCollection(TreeSet::new))));
}
```

Ejercicio 4.4

```
public Map<Nivel, Double> porcentajeCortesPorSeveridadEnRegion(String region) {
	Map<Nivel, Long> cortesPorSeveridad = cortes.stream()
		.filter(c -> c.region().equals(region))
		.collect(Collectors.groupingBy(
				CorteElectrico::severidad,
				Collectors.counting()));

	Long numCortes = cortes.stream()
			.filter(c -> c.region().equals(region))
			.count();

	return cortesPorSeveridad.keySet().stream()
		.collect(Collectors.toMap(
				x -> x,
				x -> cortesPorSeveridad.get(x)*100.0/numCortes));
}
```

Ejercicio 4.5

```
public String compañiaConMasAfectadosEnFecha(LocalDate f) {
	Map<String, Integer> consumidoresPorCompañia = cortes.stream()
		.filter(c -> c.consumidores() != null
				&& !f.isBefore(c.fechaInicio().toLocalDate())
				&& !f.isAfter(c.fechaRestablecimiento().toLocalDate()))
		.collect(Collectors.groupingBy(
				CorteElectrico::compañia,
				Collectors.summingInt(CorteElectrico::consumidores)));

		return consumidoresPorCompañia.keySet().stream()
				.max(Comparator.comparing(
					x -> consumidoresPorCompañia.get(x)))
				.get();

		// También
		return consumidoresPorCompañia.entrySet().stream()
				.max(Comparator.comparing(e -> e.getValue()))
				.get().getKey();
}
```

Test

```
public class TestInformeCortes {

	public static void main(String[] args) {

		InformeCortes inf = FactoriaCortes.leerCortes("data/power_outages.csv");
		System.out.println("Se han leido " + inf.getNumeroCortes() + " cortes");

		System.out.println("1 ====================================== ");
		testMediaAfectadosRegiones(inf, Nivel.MEDIO,
			Set.of("Southeastern Michigan"));
		testMediaAfectadosRegiones(inf, Nivel.ALTO,
			Set.of("Madison, Wisconsin", "Newark, Delaware"));

		System.out.println("2 ====================================== ");
		testCompañiasCortesMasRecientes (inf, "vandalism", 5);
		testCompañiasCortesMasRecientes (inf, "winter storm", 5);

		System.out.println("3 ====================================== ");
		testCompañiaConMayorPerdidaPorMes(inf);

		System.out.println("4 ====================================== ");
		testPorcentajeCortesPorSeveridadEnRegion (inf, "Southeastern Michigan" );
		testPorcentajeCortesPorSeveridadEnRegion (inf, "Madison, Wisconsin" );

		System.out.println("5 ====================================== ");
		testCompañiaConMasAfectadosEnFecha(inf, LocalDateTime.of(2002,8,9, 9,00));
		testCompañiaConMasAfectadosEnFecha(inf, LocalDateTime.of(2014,1,27,14,20));
	}

	private static void testCompañiaConMasAfectadosEnFecha(InformeCortes inf,
	   LocalDateTime fecha) {
		try {
			String c = inf.compañiaConMasAfectadosEnFecha(fecha);
			String msg = String.format(
				"La compañia con mas afectados en la fecha y hora %s es %s",
				fecha.toString(), c);
			System.out.println(msg);
		} catch (NoSuchElementException nse) {
			System.out.println("No se puede calcular el maximo");
		} catch (Exception e) {
			System.out.println("Excepcion inesperada");
		}
	}

	private static void testPorcentajeCortesPorSeveridadEnRegion(InformeCortes inf,
	   String region) {
		try {
			Map<Nivel, Double> m =
				inf.porcentajeCortesPorSeveridadEnRegion(region);
			String msg = String.format(
				"El porcentaje de cortes por nivel de severidad en la region
				%s es %s", region, m.toString());
			System.out.println(msg);
		} catch (Exception e) {
			System.out.println("Excepcion inesperada");
		}
	}
```

```
	private static void testCompañiaConMayorPerdidaPorMes(InformeCortes inf) {
		try {
			Map<Month, String> compañias = inf.compañiaConMayorPerdidaPorMes();
			String msg = String.format(
				"Las compañias con mayor perdida por mes son %s",
				 compañias.toString());
			System.out.println(msg);
		} catch (Exception e) {
			System.out.println("Excepcion inesperada");
		}
	}

	private static void testCompañiasCortesMasRecientes(InformeCortes inf,
	   String etiqueta, Integer n) {
		try {
			List<String> compañias =inf.compañiasCortesMasRecientes(etiqueta, n);
			String msg=String.format("Las %d compañias con la etiqueta %s son%s",
				n, etiqueta, compañias.toString());
			System.out.println(msg);
		} catch (Exception e) {
			System.out.println("Excepcion inesperada");
		}
	}

	public static void testMediaAfectadosRegiones(InformeCortes inf, Nivel n,
	   Set<String> regiones) {
		try {
			Double media = inf.mediaAfectadosRegiones(n, regiones);
			String msg = String.format(
			   "La media de los afectados de nivel %s en las regiones %s es %f",
			   n.toString(), regiones.toString(), media);
			System.out.println(msg);
		} catch (NoSuchElementException nse) {
			System.out.println("No se puede calcular la media");
		} catch (Exception e) {
			System.out.println("Excepcion inesperada");
		}
	}

}
```