Ejercicio 1

```
public record ProduccionNetflix(String titulo, Tipo tipoShow, Integer anyo,
		Duration duracion, Set<String> generos, Integer numeroTemporadas,
		Double scoreIMDB, Long popularidadIMDB)
		implements Comparable<ProduccionNetflix> {

	private static final Integer ANYO_LIMITE = 1900;

	public ProduccionNetflix {
		Checkers.check("El año de producción debe ser posterior a 1900",
			anyo > ANYO_LIMITE);
		Checkers.check("El score debe estar entre 0 y 10",
			scoreIMDB >= 0 && scoreIMDB <= 10);
		Checkers.check("La popularidad debe ser superior o igual a cero",
				popularidadIMDB >= 0);
		Checkers.check("La temporada debe concordar con el tipo de show",
				esNumeroTemporadasOK(tipoShow, numeroTemporadas));
	}

	private Boolean esNumeroTemporadasOK(Tipo tipoShow, Integer numeroTemporadas) {
		return (tipoShow.equals(Tipo.SHOW) && numeroTemporadas >= 1)
				|| (tipoShow.equals(Tipo.MOVIE) && numeroTemporadas == 0);
	}
	public int compareTo(ProduccionNetflix o) {
		int res = titulo().compareTo(o.titulo());
		if (res == 0) {
			res = anyo().compareTo(o.anyo());
		}
		return res;
	}
	public int hashCode() {
		final int prime = 31;
		int result = 1;
		result = prime * result + ((anyo == null) ? 0 : anyo.hashCode());
		result = prime * result + ((titulo == null) ? 0 : titulo.hashCode());
		return result;
	}
	public boolean equals(Object obj) {
		if (this == obj)
			return true;
		if (!(obj instanceof ProduccionNetflix))
			return false;
		ProduccionNetflix other = (ProduccionNetflix) obj;
		if (anyo == null) {
			if (other.anyo != null)
				return false;
		} else if (!anyo.equals(other.anyo))
			return false;
		if (titulo == null) {
			if (other.titulo != null)
				return false;
		} else if (!titulo.equals(other.titulo))
			return false;
		return true;
	}
}
```

Ejercicio 2

```
public class CatalogoNetflix {

	private SortedSet<ProduccionNetflix> producciones;

	public SortedSet<ProduccionNetflix> getTitulos() {
		return new TreeSet<>(producciones);
	}

	public CatalogoNetflix(Stream<ProduccionNetflix> stream) {
		Comparator<ProduccionNetflix> cmp =
			Comparator.comparing(ProduccionNetflix::anyo)
			.thenComparing(Comparator.naturalOrder());
		this.producciones = stream.collect(
			Collectors.toCollection(() -> new TreeSet<>(cmp)));
}
```

Ejercicio 3

```
public class FactoriaNetflix {

	private static final String DELIMITADOR_PRINCIPAL = ",";
	private static final String DELIMITADOR_SECUNDARIO = ";";

	public static CatalogoNetflix leeNetflix(String ruta) {

		Stream<ProduccionNetflix> sTitulos = null;
		try {
			sTitulos = Files.lines(Paths.get(ruta))
				.skip(1)
				.map(FactoriaNetflix::parse);
		} catch (IOException ioe) {
			System.err.println("Error gestionando fichero " + ruta);
		}
		return new CatalogoNetflix(sTitulos);
	}

	public static ProduccionNetflix parse(String s) {
		String[] trozos = s.replaceAll("\"", "").split(DELIMITADOR_PRINCIPAL);
		Checkers.check("Formato no válido", trozos.length == 8);

		String titulo = trozos[0].trim();
		Tipo tipo = Tipo.valueOf(trozos[1].trim());
		Integer anyo = Integer.valueOf(trozos[2].trim());
		Duration duracion = parseaDuracion(trozos[3].trim());
		Set<String> generos = parseaGeneros(trozos[4].trim());
		Integer temporadas = Integer.valueOf(trozos[5].trim());
		Double scoreImdb = Double.valueOf(trozos[6].trim());
		Long popularidadImdb = Long.valueOf(trozos[7].trim());

		return new ProduccionNetflix(titulo, tipo, anyo, duracion, generos,
			temporadas, scoreImdb, popularidadImdb);
	}

	private static Duration parseaDuracion(String cad) {
		return Duration.ofMinutes(Integer.valueOf(cad));
	}
```

```
	private static Set<String> parseaGeneros(String cad) {

		String limpia = cad.replace("[", "")
			.replace("'", "")
			.replace("]", "").replaceAll(" ", "");
		String[] trozos = limpia.split(DELIMITADOR_SECUNDARIO);

		Set<String> res = new HashSet<>();
		for (String trozo: trozos) {
			res.add(trozo);
		}
		return res;
	}

	private static Set<String> parseaGeneros2(String cad) {

		String limpia = cad.replace("[", "")
			.replace("'", "")
			.replace("]", "")
			.replaceAll(" ", "");

		return Stream.of(limpia.split(DELIMITADOR_SECUNDARIO))
			.collect(Collectors.toCollection(HashSet::new));
		}
	}
```

Ejercicio 4

Ejercicio 4.1

```
public Map<String, Set<ProduccionNetflix>> getTitulosPorGenero() {

	Map<String, Set<ProduccionNetflix>> res = new HashMap<>();

	for (ProduccionNetflix titulo: producciones) {
		for (String genero: titulo.generos()) {
			if (res.containsKey(genero)) {
				res.get(genero).add(titulo);
			} else {
				Set<ProduccionNetflix> conj = new HashSet<>();
				conj.add(titulo);
				res.put(genero, conj);
			}
		}
	}

	return res;
}
```

Ejercicio 4.2

```
public SortedSet<String> getGeneros() {

	return producciones.stream()
			.flatMap(titulo->titulo.generos().stream())
			.collect(Collectors.toCollection(TreeSet::new));
}
```

Ejercicio 4.3

```
public List<String> getTitulosDeGenerosOrdenadosPorTamanyo(List<String> generos) {

	Comparator<ProduccionNetflix> cmp = Comparator.comparing(
		(ProduccionNetflix p) -> p.generos().size())
		.thenComparing(p -> p.titulo().length());

	return getTitulos().stream()
		.filter(t -> t.generos().containsAll(generos))
		.sorted(cmp.reversed())
		.map(ProduccionNetflix::titulo)
		.collect(Collectors.toList());
   }
```

Ejercicio 4.4

```
public String getGeneroConMayorPopularidadAcumulada() {

	Map<String, Set<ProduccionNetflix>> aux = getTitulosPorGenero();

	Map<String, Long> aux2 = aux.entrySet().stream()
		.collect(Collectors.toMap(e -> e.getKey(),
					e -> popularidadAcumulada(e.getValue())));
	return aux2.entrySet().stream()
		.max(Comparator.comparing(Map.Entry::getValue)
		.get()
		.getKey();
}

private Long popularidadAcumulada(Set<ProduccionNetflix> producciones) {
	return producciones.stream()
		.mapToLong(ProduccionNetflix::popularidadIMDB)
		.sum();
}

// También
private Long popularidadAcumulada(Set<ProduccionNetflix> producciones) {
	return producciones.stream()
		 .collect(Collectors.summingLong(ProduccionNetflix::popularidadIMDB));
}
```

Ejercicio 4.5

```
public SortedMap<Integer, Double> getMediaTopNScoresDeGeneroPorAnyo (
		String genero, Integer n) {

	Map<Integer, List<ProduccionNetflix>> m = producciones.stream()
		.filter(tit -> tit.generos().contains(genero))
		.collect(Collectors.groupingBy(tit -> tit.anyo()));

	return m.entrySet().stream()
		.collect(Collectors.toMap(
			Map.Entry::getKey,
			e -> getMediaImdbNMejoresScore(e.getValue(), n),
			(e1, e2) -> e1,
			TreeMap::new
		));
}
```

```
private Double getMediaImdbNMejoresScore(List <ProduccionNetflix> cTitulos, Integer n) {

	return cTitulos.stream()
		.sorted(Comparator.comparing(ProduccionNetflix::scoreIMDB))
		.limit(n)
		.mapToDouble(ProduccionNetflix::scoreIMDB)
		.average()
		.getAsDouble();
}

// Solución alternativa
public SortedMap<Integer, Double> getMediaTopNScoresDeGeneroPorAnyo(String genero,
		Integer n) {

	return producciones.stream()
		.filter(tit -> tit.generos().contains(genero))
		.collect(Collectors.groupingBy(tit -> tit.anyo(),
			TreeMap::new,
			Collectors.collectingAndThen(Collectors.toList(),
			x -> getMediaImdbNMejoresScore (x, n))));
}
```

Ejercicio 4.6

```
public class TestNetflix {

	private static <K, V> void imprimeMap(Map<K, V> map) {
		map.entrySet().stream()
			.forEach(entry -> System.out.println( entry.getKey() + " , " +
				entry.getValue()));
	}

	public static <E> void imprimeColeccion(Collection<E> coleccion) {
		coleccion.stream()
			.forEach(x -> System.out.println(x));
	}

	public static void main(String[] args) {
		CatalogoNetflix c = FactoriaNetflix.leeNetflix("data/titulos_netflix.csv");
		System.out.println(c);
		testEjercicio1(c);
		testEjercicio2(c);
		testEjercicio3(c,List.of("comedy", "action"));
		testEjercicio4(c);
		testEjercicio5(c, "action", 5);
	}

	private static void testEjercicio1(CatalogoNetflix c) {
		System.out.println("-----------------------------------------------------");
		System.out.println("Ejercicio 4.1:");
		c.getTitulosPorGenero().entrySet().stream()
			.forEach(x -> System.out.println(x.getKey() + ": " +
				x.getValue().size()));

	}
```

```
	private static void testEjercicio2(CatalogoNetflix c) {
		System.out.println("-----------------------------------------------------");
		System.out.println("Ejercicio 4.2:");
		System.out.println(c.getGeneros());

	}

	private static void testEjercicio3(CatalogoNetflix c, List<String>generos) {
		System.out.println("-----------------------------------------------------");
		System.out.println("Ejercicio 4.3:");
		imprimeColeccion(c.getTitulosDeGenerosOrdenadosPorTamanyo(generos));
	}

	private static void testEjercicio4(CatalogoNetflix c) {
		System.out.println("-----------------------------------------------------");
		System.out.println("Ejercicio 4.4:");
		System.out.println(c.getGeneroConMayorPopularidadAcumulada());
	}

	private static void testEjercicio5(CatalogoNetflix c, String genero, Integer n) {
		System.out.println("-----------------------------------------------------");
		System.out.println("Ejercicio 4.5:");
		imprimeMap(c.getMediaTopNScoresDeGeneroPorAnyo(genero, n));

	}
}
```