Ejercicio 1

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

	private String nombre;
	private Tipo tipo;
	private String suministrador;
	private Integer demora;
	private Double precio;
	private Double descuento;
	private Set<String> combinadores;

	public Componente(String nombre, Tipo tipo, String suministrador, Integer demora,
			Double precio, Double descuento, Set<String> combinadores) {
		Checkers.check("La demora no debe superar los 45 días.",
			demora<=45);
		Checkers.check("Ningún componente se combina consigo mismo",
			!combinadores.contains(nombre));
		this.nombre=nombre;
		this.tipo=tipo;
		this.suministrador=suministrador;
		this.demora=demora;
		this.precio=precio;
		setDescuento(descuento);
		this.combinadores=new HashSet<String>(combinadores);
	}

	public String getNombre() {
		return nombre;
	}

	public Tipo getTipo() {
		return tipo;
	}

	public String getSuministrador() {
		return suministrador;
	}

	public Integer getDemora() {
		return demora;
	}

	public Double getPrecio() {
		return precio;
	}

	public void setPrecio(Double precio) {
		this.precio=precio;
	}

	public Double getDescuento() {
		return descuento;
	}

	public void setDescuento(Double descuento) {
		Checkers.check("El descuento debe estar entre 0. y 100.",
				descuento >= 0 && descuento <= 100);
		this.descuento=descuento;
	}
```

```
	public Double getPrecioDescontado() {
		return precio * (100 - descuento) / 100;
	}

	public Set<String> getCombinadores() {
		return new HashSet<String>(combinadores);
	}

	public int hashCode() {
		final int prime = 31;
		int result = 1;
		result = prime * result + ((nombre == null) ? 0 : nombre.hashCode());
		return result;
	}

	public boolean equals(Object obj) {
		if (this == obj)
			return true;
		if (obj == null)
			return false;
		if (getClass() != obj.getClass())
			return false;
		Componente other = (Componente) obj;
		if (nombre == null) {
			if (other.nombre != null)
				return false;
		} else if (!nombre.equals(other.nombre))
			return false;
		return true;
	}

	public int compareTo(Componente c) {
		return this.nombre.compareTo(c.nombre);
	}

	public String toString() {
		return "Componente [nombre=" + nombre + ", tipo=" + tipo
			+ ", suministrador=" + suministrador + ", demora=" + demora
			+ ", precio=" + precio + ", descuento=" + descuento
			+ ", combinadores=" + combinadores
			+ ", getPrecioDescontado()=" + getPrecioDescontado() + "]";
	}
}
```

Ejercicio 2

```
public class Catalogo {

	private Set<Componente> componentes;

	public Catalogo() {
		componentes = new HashSet<Componente>();
	}

	public Set<Componente> getComponentes() {
		return new HashSet<Componente>(componentes);
	}

	public void añadirComponente(Componente c) {
		componentes.add(c);
	}
```

```
	public void eliminarComponente(String n) {
		Componente b = null;
		for (Componente c: componentes) {
			if (c.getNombre().equals(n)) {
				b = c;
				break;
			}
		}

		if (b != null) {
			componentes.remove(b);
		}
	}

	public int hashCode() {
		final int prime = 31;
		int result = 1;
		result = prime * result + ((componentes == null) ? 0 : componentes.hashCode());
		return result;
	}

	public boolean equals(Object obj) {
		if (this == obj)
			return true;
		if (!(obj instanceof Catalogo))
			return false;
		Catalogo other = (Catalogo) obj;
		if (componentes == null) {
			if (other.componentes != null)
				return false;
		} else if (!componentes.equals(other.componentes))
			return false;
		return true;
	}

	public String toString() {
		return "Catalogo [componentes=" + componentes + "]";
	}
```

Ejercicio 3

```
public class FactoriaCatalogo {

	public static Catalogo leeCatalogo(String file) {
		Catalogo res = null;
		List<Componente> componentes;

		try {
			componentes = Files.readAllLines(Paths.get(file))
				.stream()
				.map(FactoriaCatalogo::parsearComponente)
				.collect(Collectors.toList());
			res = new Catalogo();
			for (Componente c: componentes) {
				res.añadirComponente(c);
			}
		} catch (IOException e) {
			e.printStackTrace();
		}

		return res;
	}
```

```
	private static Componente parsearComponente (String cadena) {
		Checkers.checkNoNull(cadena);
		String [] trozos = cadena.split(",");
		Checkers.check("Formato no válido", trozos.length == 7);

		String nombre = trozos[0].trim();
		Tipo tipo = parseaTipo(trozos[1].trim());
		String suministrador = trozos[2].trim();
		Integer demora = Integer.parseInt(trozos[3].trim());
		Double precio = Double.parseDouble(trozos[4].trim());
		Double descuento = Double.parseDouble(trozos[5].trim());
		Set<String> combinadores = parseaCombinadores(trozos[6].trim());

		return new Componente(nombre, tipo, suministrador,demora, precio, descuento,
			combinadores);
	}

	private static Set<String> parseaCombinadores(String cad) {
		String clean = cad.replace("[", "").replace("]","");
		String [] trozos = clean.split(";");

		return Arrays.stream(trozos)
			.map(String::trim)
			.filter(s->!s.isEmpty())
			.collect(Collectors.toSet());
	}

	private static Set<String> parseaCombinadores2(String cad) {
		String clean = cad.replace("[", "").replace("]","");
		String [] trozos = clean.split(";");
		Set<String> res = new HashSet<String>();

		for (String elem: trozos) {
			String el = elem.trim();
			if (!el.isEmpty()) {
				res.add(el);
			}
		}

		return res;
	}

	private static Tipo parseaTipo(String cad) {
		return Tipo.valueOf(cad.toUpperCase());
	}

}
```

Ejercicio 4

Ejercicio 4.1

```
public Map<Tipo,Long> numeroComponentesPorTipoDeSuministrador(String suministrador) {
	return componentes.stream()
		.filter(c->c.getSuministrador().equals(suministrador))
		.collect(Collectors.groupingBy(Componente::getTipo, Collectors.counting()));
}
```

Ejercicio 4.2

Solución 1

```
public Set<String> componentesConCombinadoresConPrecioMaximo(Double precio) {
	Map<String, Double> precioXComponente = componentes.stream()
		.collect(Collectors.toMap(Componente::getNombre, Componente::getPrecio));

	return componentes.stream()
		.filter(c ->
			combinadoresNoSuperanPrecio(c.getCombinadores(), precioXComponente, precio))
		.map(Componente::getNombre)
		.collect(Collectors.toSet());
}

private Boolean combinadoresNoSuperanPrecio(Set<String> combinadores,
		Map<String, Double> precioXComponente, Double precio) {

	return combinadores.stream()
		.allMatch(c -> precioXComponente.get(c) <= precio);
	}
```

Solución 2

```
public Set<String>componentesConCombinadoresConPrecioMaximo(Double precio) {
	Stream<Componente>aux = componentes.stream()
		.filter(c->combinadoresComponenteNosuperanPrecio(c,precio));

	return aux.map(Componente::getNombre).collect(Collectors.toSet());
}

private Boolean combinadoresComponenteNosuperanPrecio (Componente c, Double precio) {
	return componentesDesdeNombres(c.getCombinadores()).stream().
		allMatch(e -> e.getPrecio() <= precio);
}
```

Ejercicio 4.3

Solución 1

```
public Double precioEquipoInformatico(Collection<String> cs) {
   return componentes.stream()
		.filter(c -> cs.contains(c.getNombre()))
		.mapToDouble(Componente::getPrecioDescontado)
		.sum();
}
```

Solución 2

```
public Double precioEquipoInformatico(Collection<String> nombresComponentes) {
   return componentesDesdeNombres(nombresComponentes).stream()
		.mapToDouble(Componente::getPrecioDescontado)
		.sum();
}
```

Ejercicio 4.4

```
public List<String> combinadoresMasFrecuentesOrdenados(Integer n) {

	Map<String, Long> m = componentes.stream()
		.flatMap(c -> c.getCombinadores().stream())
		.collect(Collectors.groupingBy(Function.identity(), Collectors.counting()));

	Comparator<Map.Entry<String, Long>> c = Map.Entry.comparingByValue();

	return m.entrySet().stream()
		.sorted(c.reversed())
		.limit(n)
		.map(Map.Entry::getKey)
		.collect(Collectors.toList());
}
```

Ejercicio 4.5

Solución 1

```
public Set<String> combinacionDeCosteMinimo(Set<Tipo> ts) {
	Map<Tipo, Componente> mapMinimos = new HashMap<Tipo, Componente>();

	for (Componente c: componentes) {
		Tipo t = c.getTipo();
		if (ts.contains(t)) {
			if (!mapMinimos.containsKey(t)) {
				mapMinimos.put(t, c);
			} else {
				if (c.getPrecio() < mapMinimos.get(t).getPrecio()) {
					mapMinimos.put(t, c);
				}
			}
		}
	}

	return new HashSet(mapMinimos.values());
}
```

Solución 2

```
public Set<String> combinacionDeCosteMinimo2(Set<Tipo> ts) {
	Map<Tipo, Componente> mapMinimos = new HashMap<Tipo, Componente>();

	for (Componente c : componentes) {
		Tipo t = c.getTipo();
		if (ts.contains(t)) {
			if (mapMinimos.containsKey(t)) {
				c = minimo(c, mapMinimos.get(t));
			}
			mapMinimos.put(t, c);
		}
	}

	return new HashSet(mapMinimos.values());
}

private Componente minimo(Componente c1, Componente c2) {
	BinaryOperator<Componente> bo =
	BinaryOperator.minBy(Comparator.comparing(Componente::getPrecio));
	return bo.apply(c1, c2);
}
```

Solución 3

```
public Set<String> combinacionDeCosteMinimo3(Set<Tipo> ts) {
	Map<Tipo, List<Componente>> mapAgrupacion = agrupaComponentesFiltradosPorTipo(ts);
	Set<String> res = new HashSet<String> ();

	for (List<Componente> componentes:mapAgrupacion.values()) {
		Componente c = Collections.min(componentes,
			Comparator.comparing(Componente::getPrecio));
			res.add(c.getNombre());
		}
	return res;
}

private Map<Tipo, List<Componente>> agrupaComponentesFiltradosPorTipo(Set<Tipo> ts) {
	Map<Tipo, List<Componente>> res = new HashMap<Tipo, List<Componente>>();

	for (Componente c: componentes) {
		Tipo t = c.getTipo();
		if (ts.contains(t)) {
			if (!res.containsKey(t)) {
				List<Componente> lista = new ArrayList<Componente>();
				lista.add(c);
				res.put(t, lista);
			} else {
				res.get(t).add(c);
			}
		}
	}
	return res;
}
```

Tests

```
public class TestCatalogo {

   public static void main(String[] args) {
      Catalogo c = FactoriaCatalogo.leeCatalogo("data/catalogo.csv");
      testLeeCatalogo(c);

      //Ejercicio 4.1
      testNumeroComponentesPorTipoDeSuminstrador(c, "suministrador2");
      testNumeroComponentesPorTipoDeSuminstrador(c, "suministrador4");
      //Ejercicio 4.2
      testComponentesConCombinadoresConPrecioMaximo(c, 12.0);
      testComponentesConCombinadoresConPrecioMaximo(c, 30.0);
      //Ejercicio 4.3
      testPrecioEquipoInformatico(c, Set.of("placa1", "procesador2"));
      testPrecioEquipoInformatico(c, Set.of("placa1", "procesador2", "disco2"));
      //Ejercicio 4.4
      testCombinadoresMasFrecuentesOrdenados(c, 6);
      testCombinadoresMasFrecuentesOrdenados(c, 3);
      //Ejercicio 4.5
      Set<Tipo> tipos = Set.of(Tipo.PROCESADOR, Tipo.PLACA, Tipo.MEMORIA,
            Tipo.FUENTE, Tipo.SONIDO, Tipo.DISCO, Tipo.GRAFICA);
      testCombinacionDeCosteMinimo(c, tipos);
      Set<Tipo> tipos2 = Set.of(Tipo.PROCESADOR, Tipo.PLACA, Tipo.MEMORIA);
      testCombinacionDeCosteMinimo(c, tipos2);
   }

   private static void testLeeCatalogo(Catalogo c) {
      c.getComponentes().stream()
      .forEach(System.out::println);
   }
```

```
    private static void testNumeroComponentesPorTipoDeSuminstrador(Catalogo cat, String
            suministrador) {
        String msg = String.format(
                "\nNumero de Componentes por tipo suministrados por %s:\n %s",
                suministrador,
                cat.numeroComponentesPorTipoDeSuministrador(suministrador));
        System.out.println(msg);
    }

    private static void testComponentesConCombinadoresConPrecioMaximo(Catalogo cat,
            Double precio) {
        String msg = String.format("\nNombres de componentes cuyos combinadores no superan
                un precio igual a %.2f euros: \n %s",
                precio,
                cat.componentesConCombinadoresConPrecioMaximo(precio));
        System.out.println(msg);
    }

    private static void testPrecioEquipoInformatico(Catalogo cat, Set<String>componentes) {
        String msg = String.format ("\nPrecio de equipo informático compuesto de %s
                componentes: \n%.4f",
                componentes,
                cat.precioEquipoInformatico(componentes));
        System.out.println(msg);
    }

    private static void testCombinadoresMasFrecuentesOrdenados(Catalogo cat, Integer n) {
        String msg = String.format("\n Los %d componentes que ocurren más veces como
                combinadores: %s",
                n,
                cat.ordenarCombinadoresMasComunes(n));
        System.out.println(msg);
    }

    private static void testCombinacionDeCosteMinimo(Catalogo cat, Set<Tipo> tipos) {
        String msg = String.format("\nCombinación de coste mínimo para el conjunto de
                tipos %s:\n%s",
                tipos,
                 cat.combinacionDeCosteMinimo(tipos));
        System.out.println(msg);
    }
}
```