Ejercicio 1

```
public record Colaborador(String nombre, String departamento) implements Comparable<Colaborador> {

	public Colaborador {
		Checkers.checkNoNull(nombre, departamento);
		Checkers.check("El nombre no puede estar vacío",
					!nombre.isEmpty()); //También !nombre.isBlank()
		Checkers.check("El departamento no puede estar vacío",
					!departamento.isEmpty());
					//También !departamento.isBlank()
	}

	public int compareTo(Colaborador o) {
		int res = nombre().compareTo(c.nombre());
		if (res == 0) {
			res = departamento().compareTo(c.departamento());
		}
		return res;
	}

}
```

Ejercicio 2

```
public class Proyecto implements Comparable<Proyecto> {
	private String titulo;
	private List<Colaborador> colaboradores;
	private LocalDate fechaInicio;
	private LocalDate fechaFin;
	private SortedSet<String> ambitos;

	public Proyecto(String titulo, List<Colaborador> colaboradores,
			LocalDate fechaInicio, LocalDate fechaFin,
			SortedSet<String> ambitos) {

		Checkers.check("La fecha de fin debe ser posterior a la de inicio",
				fechaFin.isAfter(fechaInicio));
		Checkers.check("No puede haber colaboradores duplicados",
			!hayDuplicados(colaboradores));
		Checkers.checkNoNull(ambitos);
		Checkers.check("El conjunto de ámbitos no puede ser nulo ni vacío",
					!ambitos.isEmpty());

		this.titulo = titulo;
		this.colaboradores = new ArrayList<>(colaboradores);
		this.fechaInicio = fechaInicio;
		this.fechaFin = fechaFin;
		this.ambitos = new TreeSet<>(ambitos);
	}

	private Boolean hayDuplicados(List<Colaborador> colaboradores) {
		Set<Colaborador> repes = new HashSet<>(colaboradores);
		return repes.size() != colaboradores.size();
	}

	// Getters, setters y propiedades derivadas
	public String getTitulo() {
		return titulo;
	}

	public List<Colaborador> getColaboradores() {
		return new ArrayList<>(colaboradores);
	}
```

```
public LocalDate getFechaInicio() {
	return fechaInicio;
}

public void setFechaInicio(LocalDate fechaInicio) {
	Checkers.check("La fecha de inicio no puede ser posterior a la de fin",
		!fechaFin.isBefore(fechaInicio));
	this.fechaInicio = fechaInicio;
}

public LocalDate getFechaFin() {
	return fechaFin;
}

public SortedSet<String> getAmbitos() {
	return new TreeSet<>(ambitos);
}

public Colaborador getResponsable() {
	Colaborador res = null;
	if (!colaboradores.isEmpty()) {
		res = colaboradores.get(0);
	}
	return res;
}

public Long getDuracion() {
	return ChronoUnit.DAYS.between(fechaInicio, fechaFin);
	//También
	//return fechaInicio.until(fechaFin, ChronoUnit.DAYS);
	//return (long)Period.between(fechaInicio, fechaFin).getDays();
}

public Estado getEstado() {
	Estado result = null;
	if (LocalDate.now().isBefore(getFechaFin())) {
		result = Estado.CONCEDIDO;
	} else if (LocalDate.now().isAfter(getFechaFin())) {
		result = Estado.FINALIZADO;
	} else {
		result = Estado.EN_CURSO;
	}
	return result;
}

public Integer getNumeroColaboradores() {
	return colaboradores.size();
}

// Otras operaciones
public void anyadeAmbito(String ambito) {
	ambitos.add(ambito.toLowerCase());
}

public Boolean esColaborador(String nombre) {
	Boolean result = false;
	for (Colaborador c : colaboradores) {
		if (c.nombre().equalsIgnoreCase(nombre)) {
			result = true;
			break;
		}
	}
	return result;
}
```

```
	// Igualdad y orden
	public boolean equals(Object o) {
		if (this == o)
			return true;
		if (!(o instanceof Proyecto other))
			return false;
		return titulo.equals(other.titulo) &&
				getResponsable().equals(other.getResponsable());
	}

	public int hashCode() {
		return Objects.hash(titulo, getResponsable());
	}

	public int compareTo(Proyecto p) {
		int res = getTitulo().compareTo(p.getTitulo());
		if (res == 0) {
			res = getResponsable().compareTo(p.getResponsable());
		}
		return res;
	}

	public String toString() {
		return "Proyecto [titulo=" + titulo + ", colaboradores=" + colaboradores +
			", fechaInicio=" + fechaInicio
				+ ", fechaFin=" + fechaFin + ", ambitos=" + ambitos + "]";
	}
}
```

Ejercicio 3

```
public class ProyectoFinanciado extends Proyecto {

	private String entidadFinanciadora;
	private Double presupuesto;

	public ProyectoFinanciado(String titulo, List<Colaborador> colaboradores,
			LocalDate fechaInicio, LocalDate fechaFin,
			SortedSet<String> ambitos, Double presupuesto,
			String entidadFinanciadora) {
		super(titulo, colaboradores, fechaInicio, fechaFin, ambitos);
		setPresupuesto(presupuesto);
		setEntidadFinanciadora(entidadFinanciadora);
	}

	public String getEntidadFinanciadora() {
		return entidadFinanciadora;
	}

	public void setEntidadFinanciadora(String entidadFinanciadora) {
		Checkers.checkNoNull(entidadFinanciadora);
		Checkers.check("La entidad financiadora debe tener longitud > 0",
				!entidadFinanciadora.isEmpty()); // tb. isBlank()
		this.entidadFinanciadora = entidadFinanciadora;
	}

	public Double getPresupuesto() {
		return presupuesto;
	}

	public void setPresupuesto(Double presupuesto) {
		Checkers.check("El presupuesto debe ser mayor que 0", presupuesto > 0);
		Checkers.check("No mayor de 2 años si presupuesto menor de 10000",
				!(presupuesto < 10000) ||getDuracion().getYears() < 2);
		this.presupuesto = presupuesto;
	}
```

```
	public void setFechaInicio(LocalDate f) {
		Checkers.check("No mayor de 2 años si presupuesto menor de 10000",
			!(getPresupuesto() < 10000) || getDuracion().getYears() < 2);
		super.setFechaInicio(f);
	}

	public String toString() {
		return "ProyectoFinanciado [Proyecto=" + super.toString() + ",
				entidadFinanciadora=" + entidadFinanciadora
				+ ", presupuesto=" + presupuesto + "]";
	}
```

Ejercicio 4

```
private static final String SEP_PPAL = ",";
private static final String SEP_COLABORADOR = "-";
private static final String SEP_SECUNDARIO = ";";

public static Proyecto parseaProyecto(String linea) {

	Checkers.checkNoNull(linea);
	String[] partes = linea.split(SEP_PPAL);
	Checkers.check("La línea debe tener 5 ó 7 columnas",
				partes.length == 5 || partes.length == 7);
	String titulo = partes[0].trim();
	List<Colaborador> colaboradores = parseColaboradores(partes[1]);
	LocalDate fechaInicio = LocalDate.parse(partes[2].trim());
	LocalDate fechaFin = LocalDate.parse(partes[3].trim());
	SortedSet<String> ambitos = parseaAmbitos(partes[4]);
	Proyecto res = null;
	if (trozos.length == 5) {
		res = new Proyecto(titulo, colaboradores, fechaInicio, fechaFin, ambitos);
	} else if (trozos.length == 7) {
		Double presupuesto = Double.parseDouble(trozos[5].strip());
		String entidad = trozos[6].strip();
		res = new ProyectoFinanciado(titulo, colaboradores, fechaInicio, fechaFin,
				ambitos, presupuesto, entidad);
	}
	return res;
}

private static SortedSet<String> parseaAmbitos(String parte) {
	String ambitosStr = parte.trim();
	ambitosStr = ambitosStr.replace("[", "").replace("]", "");// quitar []
	String[] ambitosTokens = ambitosStr.split(SEP_SECUNDARIO);
	SortedSet<String> ambitos = new TreeSet<>();
	for (String a: ambitosTokens) {
		String amb = a.trim();
		if (!amb.isEmpty()) {
			ambitos.add(amb);
		}
	}
	return ambitos;
}

private static List<Colaborador> parseColaboradores(String parte) {
	String colabsStr = parte.trim();
	colabsStr = colabsStr.replace("[", "").replace("]", ""); // quitar []
	String[] colabsTokens = colabsStr.split(SEP_SECUNDARIO);
	List<Colaborador> colaboradores = new ArrayList<>();
	for (String colaborador: colabsTokens) {
		res.add(parseaColaborador(colaborador.strip()));
	}
	return colaboradores;
}
```

```
private static Colaborador parseaColaborador(String colaborador) {
	String[] trozos = colaborador.split(SEP_COLABORADOR);
	String msg = String.format("Formato de colaborador no válido <%s> <%d>",
			colaborador, trozos.length);
	Checkers.check(msg, trozos.length == 2);
	return new Colaborador(trozos[0].strip(), trozos[1].strip());
}
```

Ejercicio 5

Apartado 1

```
public List<String> responsablesPorAmbito(String ambito) {

	return proyectos.stream().filter(p -> p.getAmbitos().contains(ambito))
				.map(p -> p.getResponsable().nombre())
				.collect(Collectors.toList());
}
```

Apartado 2

```
public SortedMap<String, Double> presupuestoTotalPorDepartamento() {

	return proyectos.stream()
		.filter(p -> p instanceof ProyectoFinanciado)
		.map(pr -> (ProyectoFinanciado)pr)
		.collect(Collectors.groupingBy(
			p -> p.getResponsable().departamento(),
			TreeMap::new,
			Collectors.summingDouble(p -> p.getPresupuesto())));
}
```

Apartado 3

```
public SortedSet<String> titulosProyectosLargos(long nDias) {

	Map<String, Long> duracionPorTitulo = proyectos.stream()
			.collect(Collectors.groupingBy(Proyecto::getTitulo,
			Collectors.mapping(Proyecto::getDuracion,
				Collectors.collectingAndThen(
					Collectors.maxBy(Comparator.naturalOrder()),
					opt -> opt.orElse(0L)))));
	// También
	//Map<String, Long> m = proyectos.stream()
	//		.collect(Collectors.groupingBy(
	//			Proyecto::getTitulo,
	//			Collectors.collectingAndThen(
	//			Collectors.maxBy(Comparator.comparing(Proyecto::getDuracion)),
	//			opt->opt.get().getDuracion())));



	// Comparador: primero por duración desc, luego por título asc
	Comparator<String> cmp = Comparator.comparingLong(
				(String t) -> duracionPorTitulo.getOrDefault(t, 0L))
			.reversed()
			.thenComparing(Comparator.naturalOrder());

	return proyectos.stream()
		.filter(p -> p.getDuracion() > nDias).map(Proyecto::getTitulo)
		.collect(Collectors.toCollection(() -> new TreeSet<String>(cmp)));
}
```

Apartado 4

```
public String proyectoMasColaborativo() {

        Comparator<? super Proyecto> cmp = Comparator.comparing(
                (Proyecto p) -> numDepartamentosColaboradores(p))
            .thenComparing(Comparator.comparing(Proyecto::getFechaInicio)
                          .reversed());
        return proyectos.stream()
                .max(cmp).map(Proyecto::getTitulo)
                .orElse(null);
}

private Long numDepartamentosColaboradores(Proyecto proyecto) {
        Long res = proyecto.getColaboradores().stream()
                               .map(Colaborador::departamento)
                               .distinct()
                               .count();
        return res;
}
```