Ejercicio 1

```
public record Profesor(String nombre, LocalDate fechaIngreso,
                Set<Asignatura> asigsCursosAnt, Set<Asignatura> asigsPrimCuatr,
                Set<Asignatura> asigsSegCuatr, Double credTeoPrimCuatr,
                Double credTeoSegCuatr, Double credLabPrimCuatr,
                Double credLabSegCuatr, Double capacidad,
                Set<Asignatura> asigsCoordinadas) implements Comparable<Profesor> {

   public Profesor {
          //R1
          Checkers.check("Fecha de ingreso incorrecta",
             fechaIngreso.isBefore(LocalDate.now()));
          //R2
          Checkers.check("Créditos asignados incorrectos",
             credTeoPrimCuatr+credTeoSegCuatr+credLabPrimCuatr+credLabSegCuatr>=0);
          //R3
          Checkers.check("Capacidad docente no válida", capacidad > 0);
   }

   public Double getCreditosAsignados() {
          return credTeoPrimCuatr+credTeoSegCuatr + credLabPrimCuatr+credLabSegCuatr;
   }

   public Double getCreditosTeoricos() {
          return credTeoPrimCuatr() + credTeoSegCuatr();
   }

   public Integer getExperiencia() {
          Period res = Period.between(LocalDate.now(), fechaIngreso);
          return res.getYears();
   }

   public Boolean esOcioso() {
          return getCreditosAsignados() < (capacidad/2.0);
   }

   public Boolean esCoordinador() {
          return !asigsCoordinadas().isEmpty();
   }

   public Set<Asignatura> asignaturasImpartidas(){
          Set<Asignatura> res = new HashSet<>(asigsPrimCuatr());
          res.addAll(asigsSegCuatr());
          return res;
   }
```

```java
    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 instanceof Profesor))
            return false;
        Profesor other = (Profesor) obj;
        if (nombre == null) {
            if (other.nombre != null)
                return false;
        } else if (!nombre.equals(other.nombre))
            return false;
        return true;
    }

    public int compareTo(Profesor p) {
        return nombre().compareTo(p.nombre());
    }

}
```

Ejercicio 2

```
public class Departamento {

	private Set<Profesor> profesores;

	public Departamento() {
		profesores = new HashSet<Profesor>();
	}

	public Departamento(Stream<Profesor> profesores) {
		this.profesores = profesores.collect(Collectors.toSet());
	}

	public Set<Profesor> getProfesores() {
		return new HashSet<Profesor> (profesores);
	}

	public int hashCode() {
		return Objects.hash(profesores);
	}

	public boolean equals(Object obj) {
		if (this == obj)
			return true;
		if (obj == null)
			return false;
		if (getClass() != obj.getClass())
			return false;
		Departamento other = (Departamento) obj;
		return Objects.equals(profesores, other.profesores);
	}

	public String toString() {
		return "Departamento [profesores=" + profesores + "]";
	}

	public void añadirProfesor(Profesor p) {
		profesores.add(p);
	}

	public void eliminaProfesor(String nombre) {
		Profesor pb = null;
		for(Profesor p:profesores) {
			if (p.nombre().equals(nombre)) {
				pb = p;
				break;
			}
		}
		if (pb != null) {
			profesores.remove(pb);
		}
	}

}
```

Ejercicio 3

```
public class FactoriaDepartamento {

	public static Departamento leerDepartamento(String nombreFichero) {
		Departamento res = null;
		try {
			Stream<Profesor> stProf =
					Files.readAllLines(Paths.get(nombreFichero))
					.stream()
					.map(FactoriaDepartamento::parsearProfesor);
			res = new Departamento(stProf);
		} catch (IOException ioe) {
			System.err.println("Error leyendo fichero");
		}
		return res;

	}

	public static Profesor parsearProfesor(String cad) {
		Checkers.checkNoNull(cad);
		String trozos[] = cad.split(",");
		Checkers.check("Formato no válido <" + cad + ">", trozos.length == 11);
		String nombre = trozos[0].trim();
		LocalDate fechaIngreso = parseaFecha(trozos[1].trim());
		Set<Asignatura> asigCursosAnts = parseaConjAsig(trozos[2].trim());
		Set<Asignatura> asigPrimCuat = parseaConjAsig(trozos[3].trim());
		Set<Asignatura> asigSegCuat = parseaConjAsig(trozos[4].trim());
		Double credTeoPrimCuat = Double.parseDouble(trozos[5].trim());
		Double credTeoSegCuat = Double.parseDouble(trozos[6].trim());
		Double credLabPrimCuat = Double.parseDouble(trozos[7].trim());
		Double credLabSegCuat = Double.parseDouble(trozos[8].trim());
		Double capacidad = Double.parseDouble(trozos[9].trim());
		Set<Asignatura> asigCoord = parseaConjAsig(trozos[10].trim());

		return new Profesor(nombre, fechaIngreso, asigCursosAnts, asigPrimCuat,
				asigSegCuat, credTeoPrimCuat, credTeoSegCuat,
				credLabPrimCuat, credLabSegCuat, capacidad, asigCoord);
	}

	private static Set<Asignatura> parseaConjAsig(String cadConjAsig) {
		String limpia = cadConjAsig.replace("{", "").replace("}", "").trim();
		Set<Asignatura> res = new HashSet<Asignatura>();
		String[] trozos = limpia.split(";");
		for (int i = 0; i < trozos.length; i++) {
			if (!trozos[i].trim().isEmpty()) {
				Asignatura a = Asignatura.valueOf(trozos[i].trim());
				res.add(a);
			}
		}

		return res;
	}

	private static LocalDate parseaFecha(String cadFecha) {
		return LocalDate.parse(cadFecha, DateTimeFormatter.ofPattern("d/M/yyyy"));
	}
}
```

Ejercicio 4

Ejercicio 4.1

```
public Map<Asignatura, List<Profesor>> profesoresPorAsignatura() {
	Map<Asignatura, List<Profesor>> res = new HashMap<Asignatura, List<Profesor>>();
		for (Profesor p: profesores) {
			for (Asignatura a: p.asignaturasImpartidas()) {
				if (res.containsKey(a)) {
					res.get(a).add(p);
				} else {
					List<Profesor> profesores = new ArrayList<Profesor>();
					profesores.add(p);
					res.put(a, profesores);
				}
			}

		}
	return res;
}
```

Ejercicio 4.2

```
Set<Profesor> profesoresQueSoloImpartenAsignaturasQueCoordinan() {
	return profesores.stream()
		.filter(p -> p.asigsCoordinadas().containsAll(p.asignaturasImpartidas()))
		.collect(Collectors.toSet());
}
```

Ejercicio 4.3

```
public Boolean departamentoResponsable() {
	return profesores.stream()
		.filter(Profesor::esCoordinador)
		.allMatch(p -> p.getExperiencia() >= 5 &&
			p.asignaturasImpartidas().containsAll(p.asigsCoordinadas()));
}
```

Ejercicio 4.4

```
public List<String> ordenarProfesoresPorNumeroCreditosTeoricos() {
	Comparator<Profesor> c = Comparator.comparing(Profesor::getCreditosTeoricos)
		.reversed();

	return profesores.stream()
		.filter(p -> p.getCreditosTeoricos()>0)
		.sorted(c)
		.map(Profesor::nombre)
		.collect(Collectors.toList());
}
```

Ejercicio 4.5

```
public Integer añoIncorporacionMasCoordinadores() {
	Map<Integer, Long> m = profesores.stream()
		.filter(Profesor::esCoordinador)
		.collect(Collectors.groupingBy(p -> p.fechaIngreso().getYear(),
			Collectors.counting()));

	return m.entrySet().stream()
		.max(Map.Entry.comparingByValue())
		.map(Map.Entry::getKey)
		.get();
	}
```

Ejercicio 4.6

```
public class TestDepartamento {

	public static void main(String[] args) {
		Departamento dep =
			FactoriaDepartamento.leerDepartamento("data/profesores.csv");
		dep.getProfesores().stream()
			.forEach(System.out::println);

		System.out.println("1 ============================");
		testProfesoresPorAsignatura(dep);

		System.out.println("2 ============================");
		testProfesoresQueSoloImpartenAsignaturasQueCoordinan(dep);

		System.out.println("3 ============================");
		testDepartamentoResponsable(dep);

		System.out.println("4 ============================");
		testOrdenarProfesoresPorNumeroCreditosTeoricos(dep);

		System.out.println("5 ============================");
		testAñoIncorporacionMasCoordinadores(dep);

	}

	private static void testProfesoresPorAsignatura(Departamento dep) {
		try {
		   Map<Asignatura, List<Profesor>> m = dep.profesoresPorAsignatura();
		   m.entrySet().stream()
		       .forEach(e->System.out.println(e.getKey()+"-->"+e.getValue()));
		} catch (Exception e) {
		   System.err.println("Capturada excepción inesperada");
		}
	}
```

```
    private static void testProfesoresQueSoloImpartenAsignaturasQueCoordinan
    (Departamento dep) {
           try {
              Set<Profesor> sp =
                 dep.profesoresQueSoloImpartenAsignaturasQueCoordinan();
              System.out.println(
                 "Los profesores que solo imparten las asign. que coordinan son:");
              sp.stream()
                 .forEach(System.out::println);
           } catch (Exception e) {
              System.err.println("Capturada excepción inesperada");
           }
    }


    private static void testDepartamentoResponsable(Departamento dep) {
           try {
              Boolean res = dep.departamentoResponsable();
              String msg = String.format("¿Es el departamento responsable? %s",
                 res.toString());
              System.out.println(msg);
           } catch (Exception e) {
              System.err.println("Capturada excepción inesperada");
           }
    }

    private static void testOrdenarProfesoresPorNumeroCreditosTeoricos
    (Departamento dep) {
           try {
              List<String> sp = dep.ordenarProfesoresPorNumeroCreditosTeoricos();
              System.out.println("Profesores ordenados por numero créditos teóricos");
              sp.stream()
                 .forEach(System.out::println);
           } catch (Exception e) {
              System.err.println("Capturada excepción inesperada");
           }
    }

    private static void testAñoIncorporacionMasCoordinadores(Departamento dep) {
           try {
              Integer a = dep.añoIncorporacionMasCoordinadores();
              String msg = String.format(
                 "El año en el que se incorporaron más coordinadores fue %d", a);
              System.out.println(msg);
           } catch (Exception e) {
              System.err.println("Capturada excepción inesperada");
           }
    }

}
```