OBJETIVO

Este documento contiene material de apoyo para el boletín T7 del trabajo práctico. En concreto, el documento incluye la parte del tipo Profesor que se trabajará en el aula. El profesor la explicará en la sesión de laboratorio y responderá a sus dudas al respecto. El alumno, por su parte, debe añadir el código a su proyecto de curso, y realizar por su cuenta el resto de ejercicios del boletín.

INTERFACES

Interfaz `Profesor`

```
public interface Profesor extends Persona {
	// Añadir a los métodos que ya había:
	List<Asignatura> getAsignaturas();
	List<Double> getCreditos();
	// TODO: Double getDedicacionTotal();
	void imparteAsignatura(Asignatura asig, Double dedicacion);
	Double dedicacionAsignatura(Asignatura asig);
	void eliminaAsignatura(Asignatura asig);
}
```

CLASES

Clase `ProfesorImpl`

```
public class ProfesorImpl extends PersonaImpl implements Profesor {
	// Sólo se muestra lo que hay que añadir para el boletín T7

	private List<Asignatura> asignaturas;
	private List<Double> creditos;

	public ProfesorImpl(String dni, String nombre, String apellidos,
			LocalDate fechaNacimiento, String email, Categoria
			categoria) {
		// ...
		asignaturas = new ArrayList<Asignatura>();
		creditos = new ArrayList<Double>();
	}

	public ProfesorImpl(String dni, String nombre, String apellidos,
			LocalDate fechaNacimiento, String email, Categoria
			categoria, Departamento departamento) {
		// ...
		asignaturas = new ArrayList<Asignatura>();
		creditos = new ArrayList<Double>();
	}

	// Devolvemos copias de las propiedades agregadas
	public List<Asignatura> getAsignaturas() {
		return new ArrayList<Asignatura>(asignaturas);
	}

	public List<Double> getCreditos() {
		return new ArrayList<Double>(creditos);
	}
```

Clase `ProfesorImpl` (continúa…)

```
	public void imparteAsignatura(Asignatura asig, Double dedicacion) {
		checkAsignaturaDepartamento(asig);
		checkCreditosAsignatura(asig, dedicacion);
		if (asignaturas.contains(asig)) {
			// El profesor ya imparte esta asignatura:
			// hay que actualizar la dedicación
			actualizaDedicacion(asig, dedicacion);
		} else {
			// El profesor no imparte aún esta asignatura:
			// hay que añadirla a ambas listas
			nuevaAsignatura(asig, dedicacion);
		}
	}

	private void checkAsignaturaDepartamento(Asignatura asig) {
		if (getDepartamento() == null ||
		    !getDepartamento().getAsignaturas().contains(asig)) {
			throw new ExcepcionProfesorOperacionNoPermitida(
				"Un profesor no puede impartir una asignatura "
				+ "de otro departamento.");
		}
	}

	private void checkCreditosAsignatura(Asignatura asig, Double dedicacion){
		if (dedicación <= 0 || asig.getCreditos() < dedicacion) {
			throw new ExcepcionProfesorOperacionNoPermitida(
				"La dedicación debe ser mayor que 0 y menor o igual "
				+ "que el número de créditos de la asignatura.");
		}
	}

	// Modifica la posición correspondiente de la lista de dedicaciones
	private void actualizaDedicacion(Asignatura asig, Double nuevaDedicacion) {
		int posicionAsignatura = asignaturas.indexOf(asig);
		creditos.set(posicionAsignatura, nuevaDedicacion);
	}

	// Añadimos la asignatura y la dedicación al final de sendas listas
	private void nuevaAsignatura(Asignatura asig, Double dedicacion) {
		asignaturas.add(asig);
		creditos.add(dedicacion);
	}

	// Eliminamos la asignatura de la lista, y la posición correspondiente
	// de la lista de dedicaciones
	public void eliminaAsignatura(Asignatura asig) {
		int posicionAsignatura = asignaturas.indexOf(asig);
		if (posicionAsignatura >= 0) {
			creditos.remove(posicionAsignatura);
			asignaturas.remove(asig);
		}
	}

	// Devolvemos el valor almacenado en la posición correspondiente de la
	// lista de dedicaciones, o 0 si la asignatura no es impartida
	public Double dedicacionAsignatura(Asignatura asig) {
		Double res = 0.0;
		int posicionAsignatura = asignaturas.indexOf(asig);
		if (posicionAsignatura >= 0) {
			res = creditos.get(posicionAsignatura);
		}
		return res;
	}
}
```

TESTS

La clase `TestProfesor` se proporciona en un fichero independiente.