BUILDER PATTERN: CASO DE USO

TESTS DE CLASES DE DOMINIO

```
public class Alumno  {
    . . .
    public boolean puedeTomarCurso (Curso curso) {
        for (Curso requirement: curso.getRequisitos()) {
            if (! haSuperado(requirement)) {
                return false;
            }
        }
        return true;
    }

    private boolean haSuperado(Curso curso) {
        for (Registro registro: registros) {
            if (registro.getCurso().equals(curso) &&
                Boolean.TRUE.equals(registro.getSupera())) {
                return true;
            }
        }
        return false;
    }

}
```

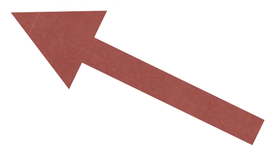
Queremos un unit-test para este método

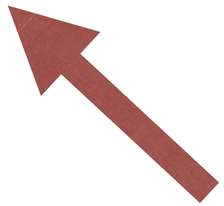
Devuelve true si ha superado todos los requisitos del curso

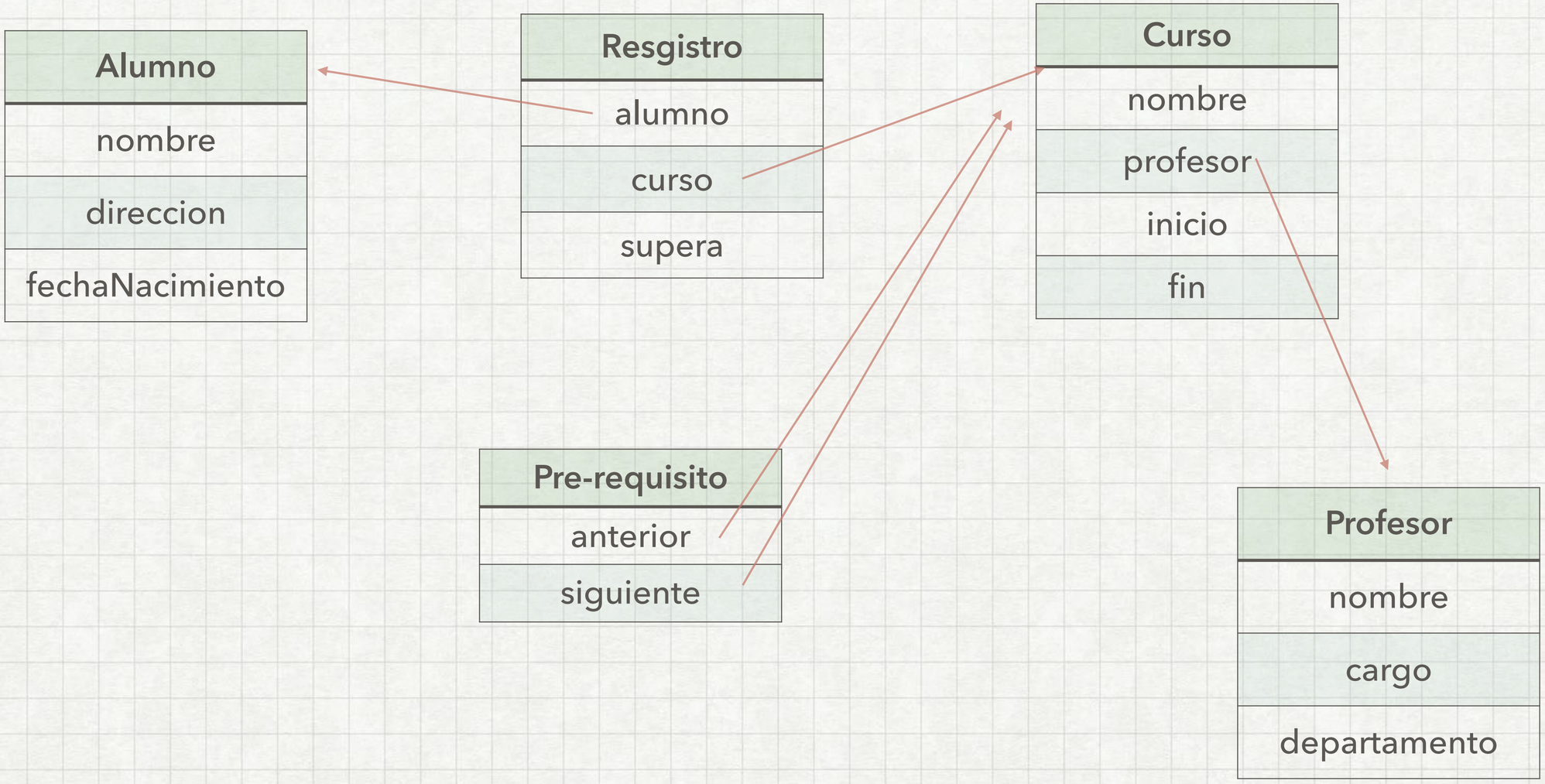
Devuelve true si ha superado un determinado curso

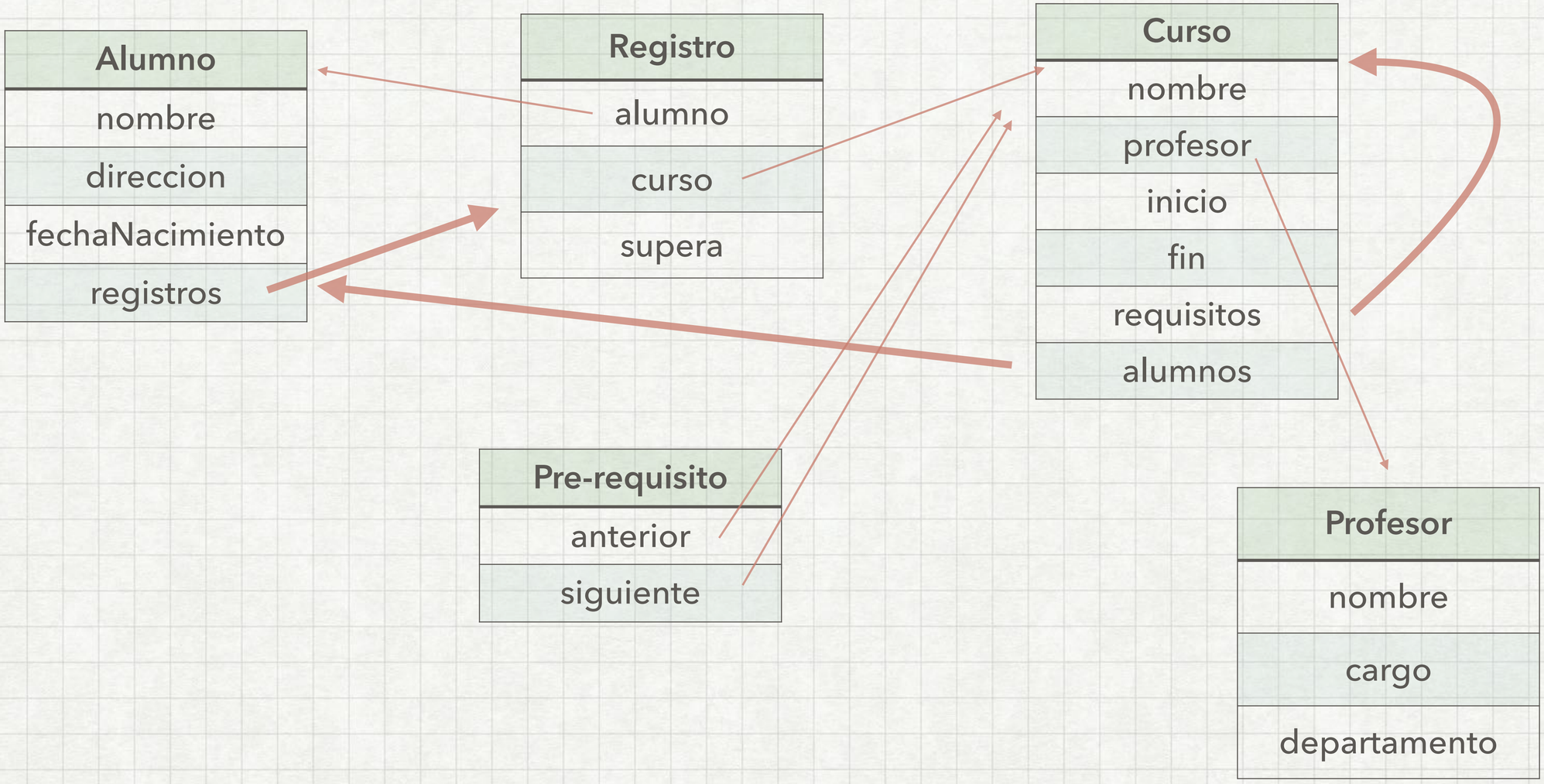
```
public class Alumno  {
    . . .

    public boolean puedeTomarCurso (Curso curso) {
        for (Curso requirement: curso.getRequisitos()) {
            if (! haSuperado(requirement)) {
                return false;
            }
        }
        return true;
    }

    private boolean haSuperado(Curso curso) {
        for (Registro registro: registros) {
            if (registro.getCurso().equals(curso) &&
                Boolean.TRUE.equals(registro.getSupera())) {
                return true;
            }
        }
        return false;
    }

}
```

Queremos un unit-test para este método






Alumno
nombre
direccion
fechaNacimiento
Resgistro
alumno
curso
supera
Curso
nombre
profesor
inicio
fin
Pre-requisito
anterior
siguiente
Profesor
nombre
cargo
departamento


Alumno
nombre
direccion
fechaNacimiento
registros
Registro
alumno
curso
supera
Curso
nombre
profesor
inicio
fin
requisitos
alumnos
Pre-requisito
anterior
siguiente
Profesor
nombre
cargo
departamento

PROBLEMA: CONSTRUIR LOS OBJETOS NECESARIOS

```
@Test
public void test_con_requisitos() {

    Curso patterns1 = new Curso();
    Curso patterns2 = new Curso();
    patterns1.setNombre("patterns 1");
    patterns2.setNombre("patterns 2");

    Requisito requisito = new Requisito();
    requisito.setAnterior(patterns1);
    requisito.setSiguiente(patterns2);
    patterns2.nuevoRequisito(patterns1);

    Alumno alumno = new Alumno();
    alumno.setNombre("Juana");


    Registro registroPatterns1 = new Registro();
    registroPatterns1.setCurso(patterns1);
    registroPatterns1.setAlumno(alumno);
    registroPatterns1.setSupera(true);
    alumno.getRegistros().add(registroPatterns1);
    patterns1.nuevoAlumno(alumno);

    assertTrue(alumno.puedeTomarCurso(patterns2));
}
```

patterns 1 es pre-requisito para el curso patterns 2

El alumno ha superado patterns1

test:

- El alumno puede cursar patterns 2

PROBLEMA: CONSTRUIR LOS OBJETOS NECESARIOS

```
@Test
public void test_con_requisitos() {

    Curso patterns1 = new Curso();
    Curso patterns2 = new Curso();
    patterns1.setNombre("patterns 1");
    patterns2.setNombre("patterns 2");

    Requisito requisito = new Requisito();
    requisito.setAnterior(patterns1);
    requisito.setSiguiente(patterns2);
    patterns2.nuevoRequisito(patterns1);

    Alumno alumno = new Alumno();
    alumno.setNombre("Juana");


    Registro registroPatterns1 = new Registro();
    registroPatterns1.setCurso(patterns1);
    registroPatterns1.setAlumno(alumno);
    registroPatterns1.setSupera(true);
    alumno.getRegistros().add(registroPatterns1);
    patterns1.nuevoAlumno(alumno);

    assertTrue(alumno.puedeTomarCurso(patterns2));
}
```

Preparación

Test

USANDO UN BUILDER

```
@Test
public void test_con_requisitos() {

    Alumno alumno = new Alumno();
    alumno.setNombre("Juana");

    Curso patterns1 = new CursoBuilder("patterns 1").addAlumno(alumno, true).build();
    Curso patterns2 = new CursoBuilder("patterns 2").addRequisito(patterns1).build();

    assertTrue(alumno.puedeTomarCurso(patterns2));
}
```

aprobado

- Sin redundancias
- Legible

EL BUILDER

```java
package edu.pattern.builder;

public class CursoBuilder {

    private Curso curso;

    public CursoBuilder (String nombre) {
        curso = new Curso();
        curso.setNombre(nombre);
        curso.setProfesor(new Profesor());
    }

    public CursoBuilder addAlumno(Alumno alumno, boolean supera) {

        Registro registro = new Registro();
        registro.setCurso(curso);
        registro.setAlumno(alumno);
        registro.setSupera(supera);

        alumno.addRegistro(registro);
        curso.nuevoAlumno(alumno);

        return this;
    }

    public CursoBuilder addRequisito (Curso previo) {
        Requisito requirement = new Requisito();
        requirement.setAnterior(previo);
        requirement.setSiguiente(curso);

        curso.nuevoRequisito(previo);

        return this;
    }

    public Curso build() {
        return curso;
    }
}
```

valores por defecto si son necesarios

fluent API para tener un código expresivo

creación de los objetos y propiedades básicas

y de las redundancias

```java
@Test
public void test_sin_requisitos() {

    Alumno alumno = new Alumno();
    alumno.setNombre("Juana");

    Curso patterns1 = new CursoBuilder("patterns 1").build();
    Curso patterns2 = new CursoBuilder("patterns 2").addRequisito(patterns1).build();

    assertFalse(alumno.puedeTomarCurso(patterns2));
}

@Test
public void test_con_requisitos() {

    Alumno alumno = new Alumno();
    alumno.setNombre("Juana");

    Curso patterns1 = new CursoBuilder("patterns 1").addAlumno(alumno, true).build();
    Curso patterns2 = new CursoBuilder("patterns 2").addRequisito(patterns1).build();

    assertTrue(alumno.puedeTomarCurso(patterns2));
}

@Test
public void test_con_requisitos_no_superados() {

    Alumno alumno = new Alumno();
    alumno.setNombre("Juana");

    Curso patterns1 = new CursoBuilder("patterns 1").addAlumno(alumno, false).build();
    Curso patterns2 = new CursoBuilder("patterns 2").addRequisito(patterns1).build();

    assertFalse(alumno.puedeTomarCurso(patterns2));
}
```

No ha hecho patterns 1

Ha aprobado patterns 1

Ha suspendido patterns 1