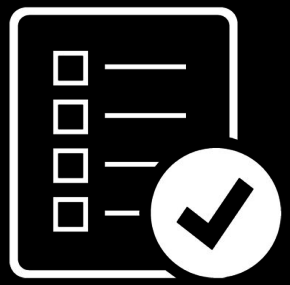
Programming 5

Setup and Refactoring

KdG Karel de Grote Hogeschool



- **Docker**
- **Schema creation and seeding**
- **Modeling many to many**

Docker

- Install Docker Desktop (and Docker Compose)
- We'll use Docker to host our **development database** consistently across different machines
  - Our application will continue to run locally on the host machine.

This is not a course on Docker. Docker will be taught in one of your Infrastructure courses.

Docker

docker-compose.yml

```yaml
services:
    db:
        image: postgres:17.2-alpine
        restart: always
        environment:
            POSTGRES_DB: 'app'
            POSTGRES_USER: 'spring'
            POSTGRES_PASSWORD: 'spring'
        ports:
            - '5432:5432'
        volumes:
            - db-data:/var/lib/postgresql/data
volumes:
    db-data:
```

Update accordingly …

Docker

- Update your **`application.properties`**

```
spring.datasource.url=jdbc:postgresql://localhost:5432/app
spring.datasource.username=spring
spring.datasource.password=spring
```

Check these specifically

```
spring.datasource.driver-class-name
```

Remove this setting, especially if it refers to H2!

Connect to the database in IntelliJ

developers

3 rows | Tx: Auto | DDL | CSV

WHERE | ORDER BY

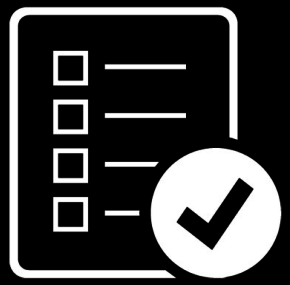
	id	email	name
1	1	raoul.vandenberge@kdg.be	Raoul Van den Berge
2	2	lars.willemsens@kdg.be	Lars Willemsens
3	3	linus@linuxfoundation.org	Linus Torvalds

Database

DDL

- issuemanagement@localhost 1 of 2
  - issuemanagement 1 of 3
    - public
      - tables 3
        - developers
        - issue_assignments
        - issues
      - sequences 2
    - Database Objects
  - Server Objects

If your application is unable to connect to your database, connecting through IntelliJ is a good first step in troubleshooting!



- Docker
- Schema creation and seeding
- Modeling many to many

Seeding with SQL

- Ensure a code-first approach
  - The database schema will be created based on your Java code
- Remove **`schema.sql`**
- Update **`application.properties`**

```
spring.jpa.generate-ddl=true
spring.jpa.hibernate.ddl-auto=create
```

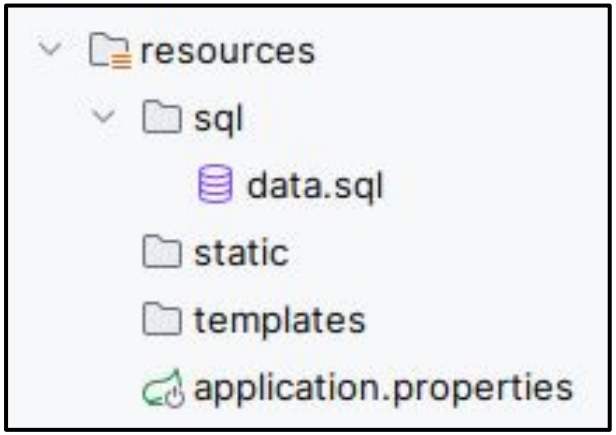
Seeding with SQL

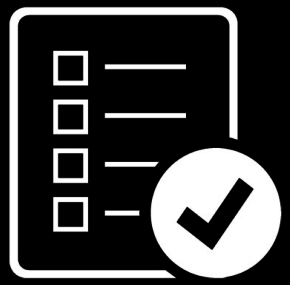
- Update your configuration so that your data comes from an SQL file:

```
spring.sql.init.data-locations=classpath:sql/data.sql
spring.sql.init.mode=always

spring.jpa.defer-datasource-initialization=true
```

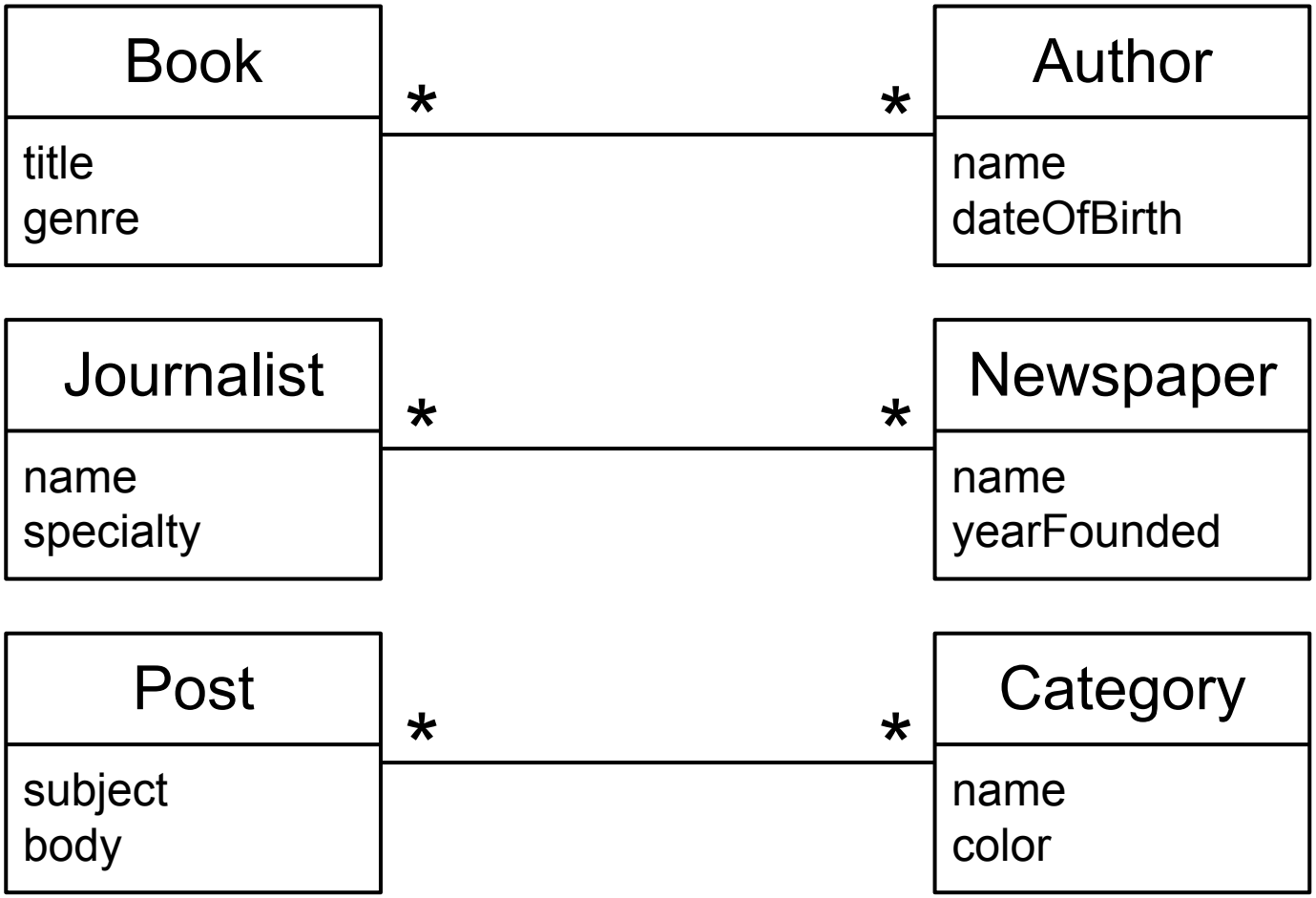
- [Quick Guide on Loading Initial Data with Spring Boot]



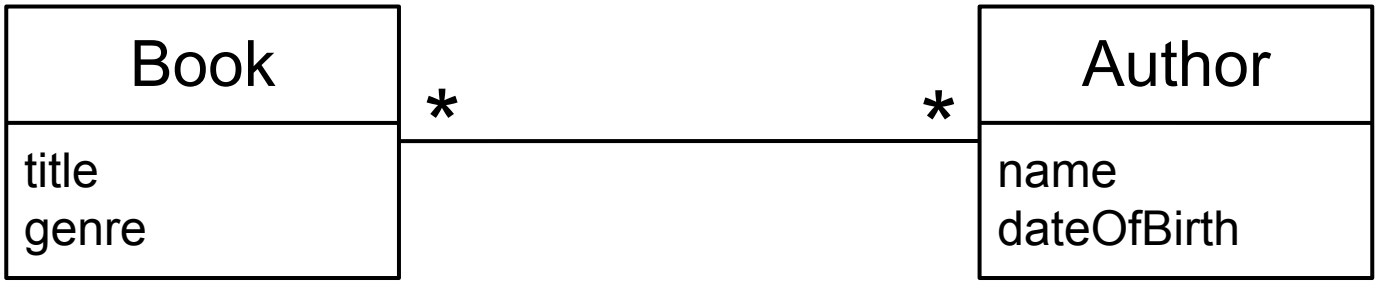


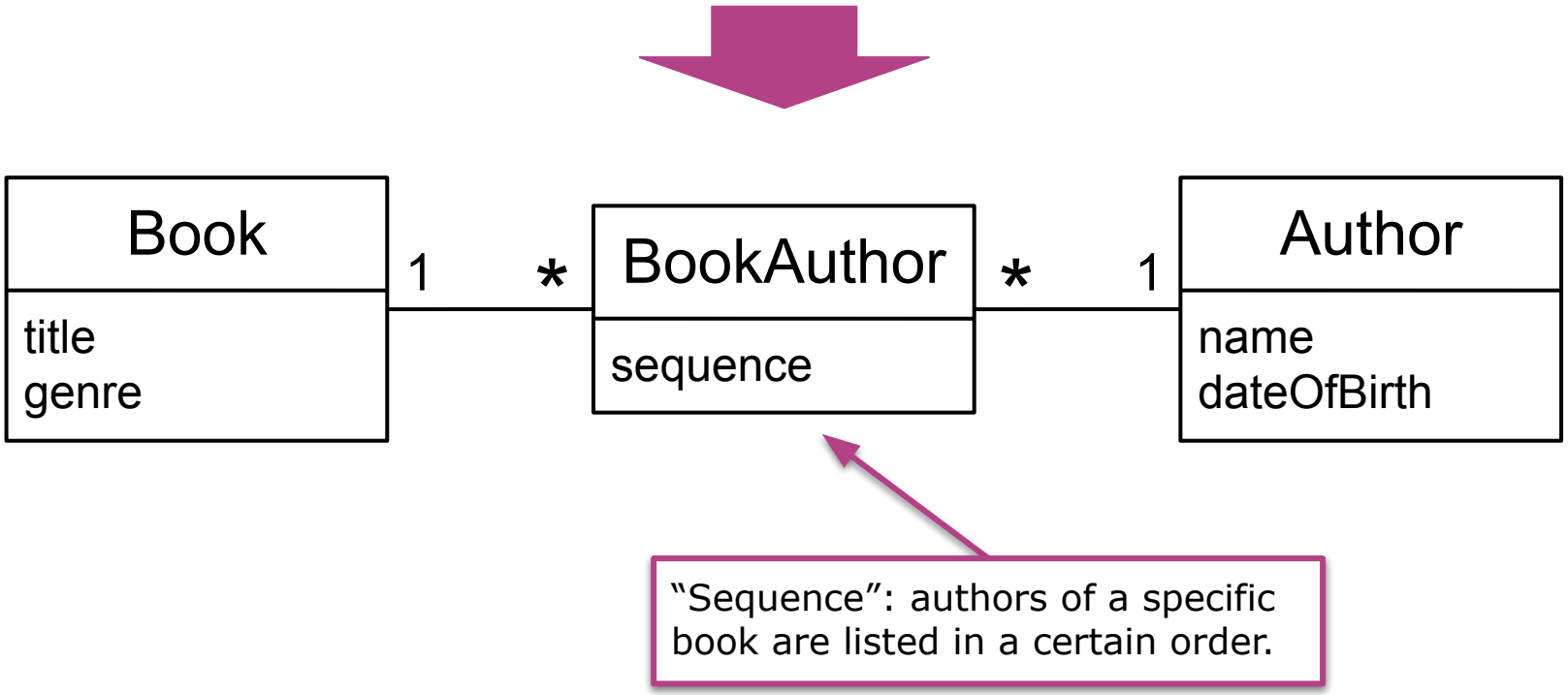
- Docker
- Schema creation and seeding
- **Modeling many to many**

Many to many

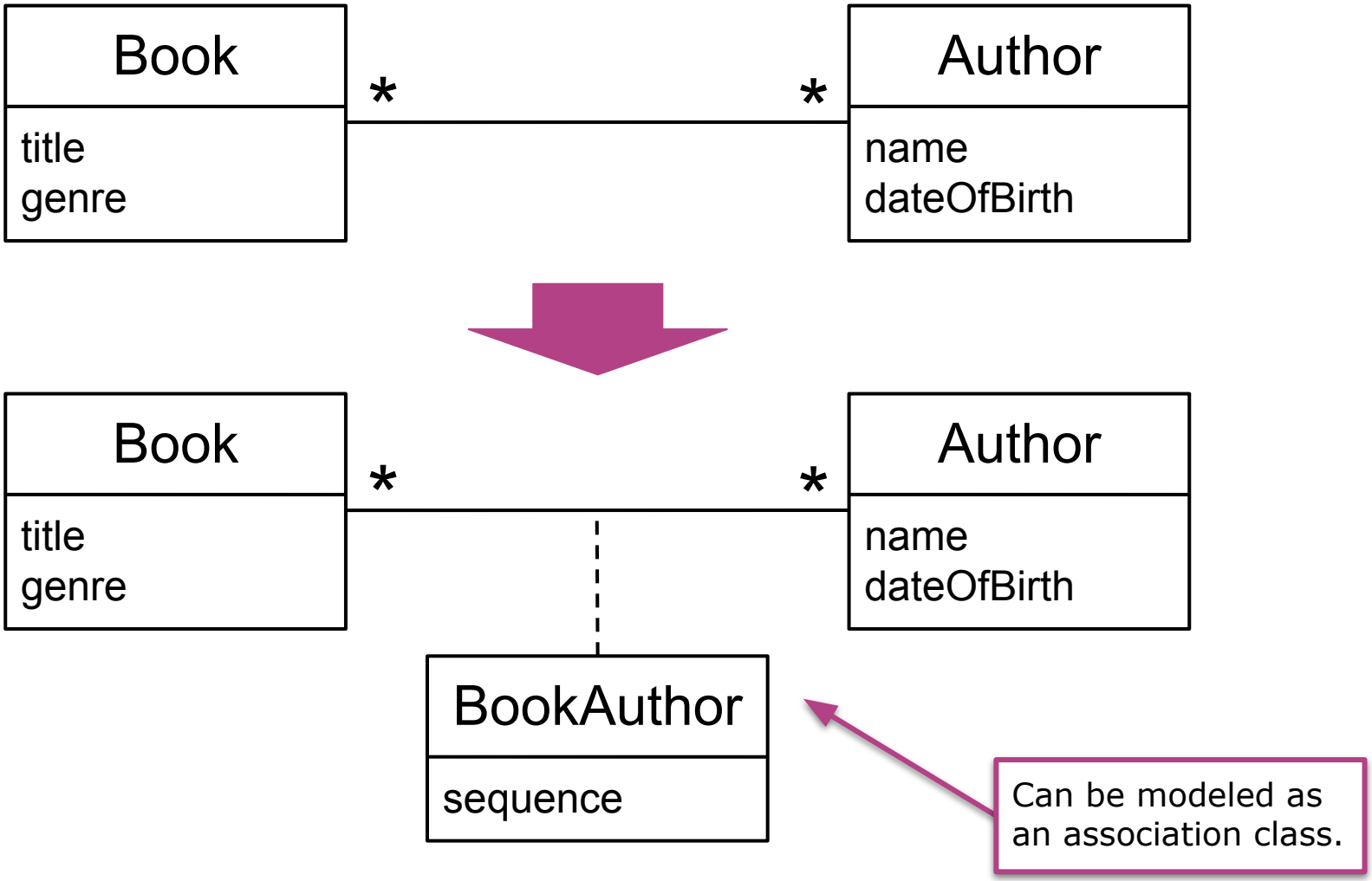

Book
title
genre
*
*
Author
name
dateOfBirth
Journalist
name
specialty
*
*
Newspaper
name
yearFounded
Post
subject
body
*
*
Category
name
color

Many to many - first scenario





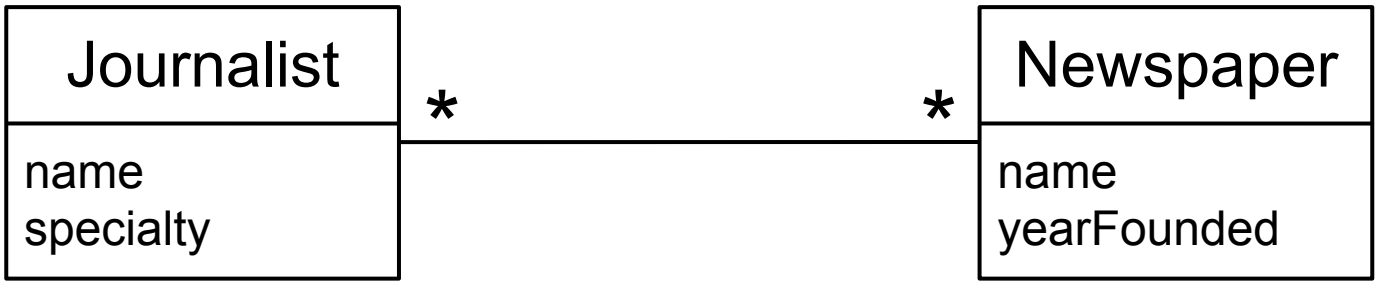
Many to many - first scenario

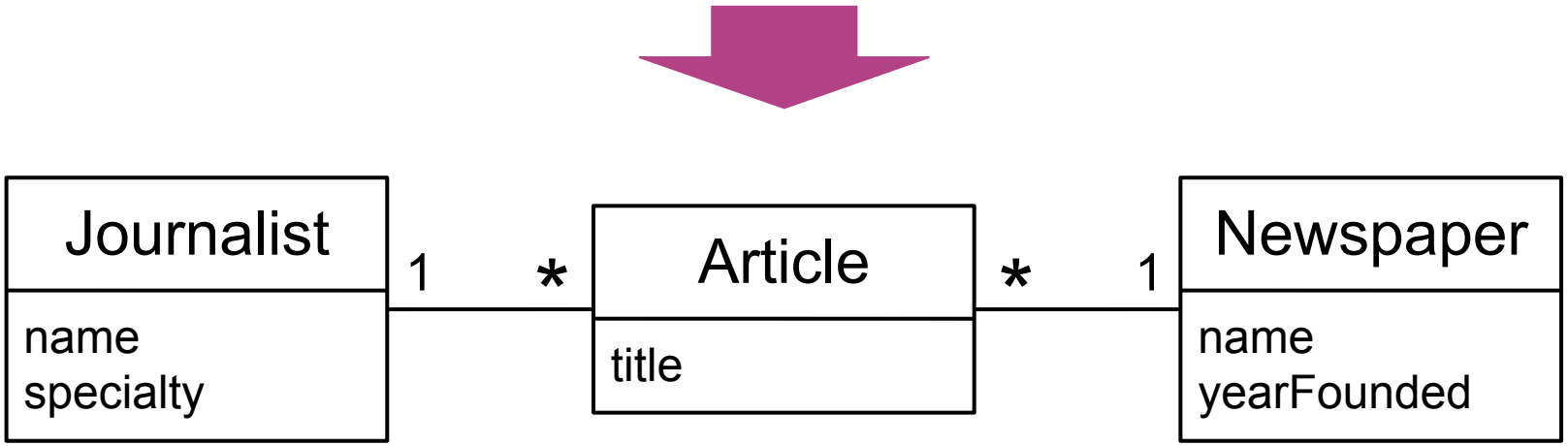


Many to many - first scenario

- There is **information** to be kept about this many to many association
  - This will **almost always** be the case!
- The association is **<u>unique</u>**
  - There can be only one association between any two specific objects
- The class needs to be modeled as part of the domain (full class or association class)
- A name such as ‘BookAuthor’ can be considered okay

Many to many - second scenario



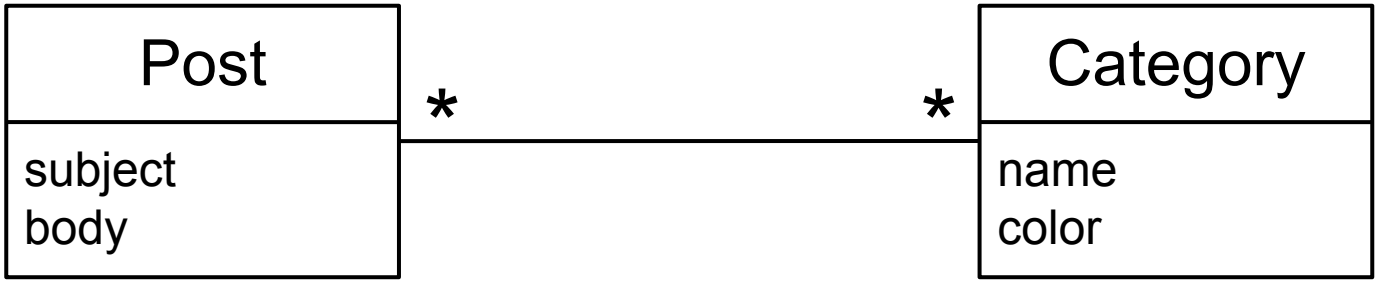


For **this** example we want to model this as 'Article'. Full class 'Contract' would be **another** possible association between Journalist and Newspaper!

Many to many - second scenario

- There is **information** to be kept about this many to many association
  - This will **almost always** be the case!
- The association is **<u>not unique</u>**
  - There can be multiple associations between two specific objects
- The class needs to be modeled as part of the domain as a **full class**
- You must come up with a **meaningful name** for the class (*not* ‘BookAuthor’)

Many to many - third scenario



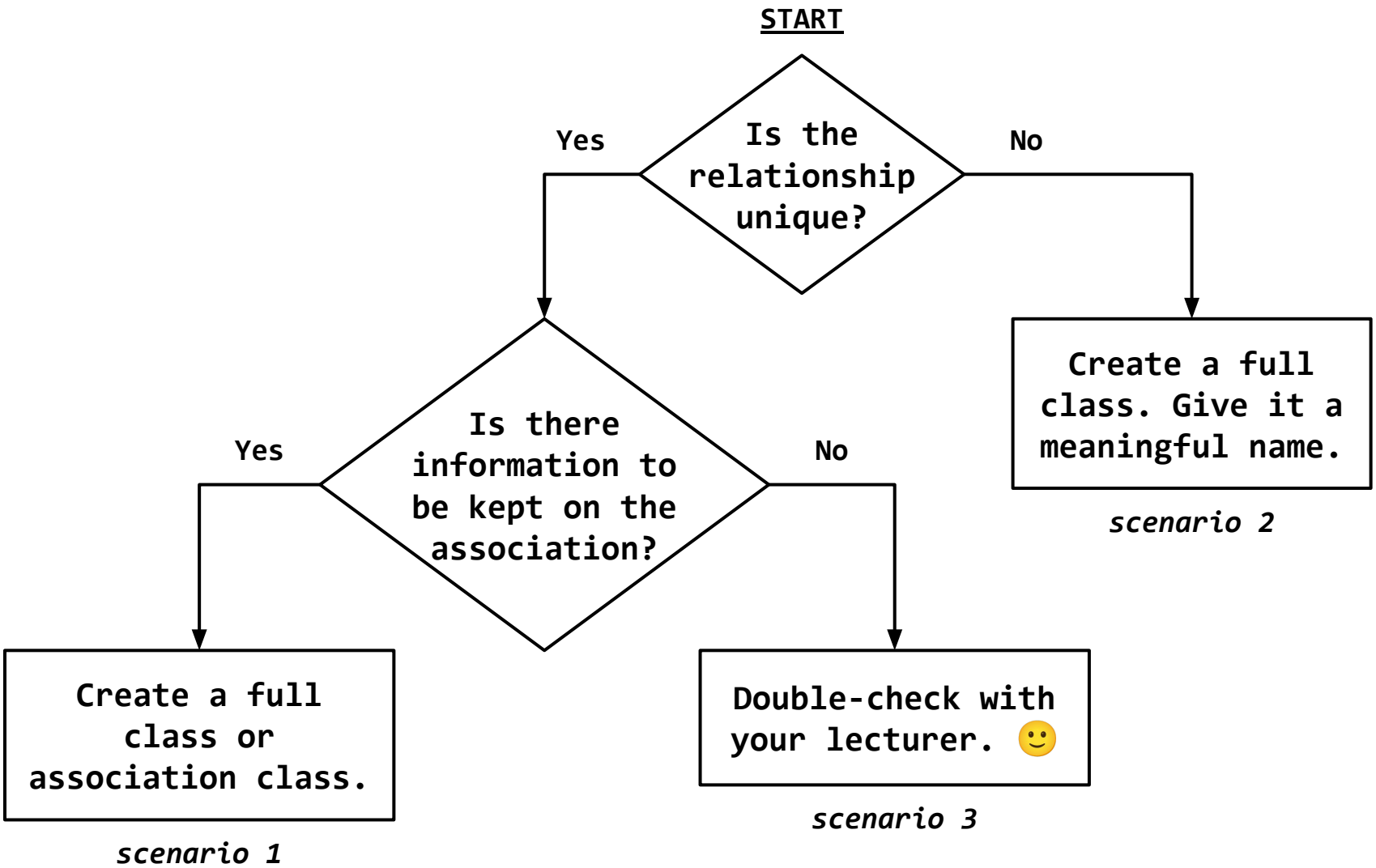
Modeled as-is!
In the domain, there's **no** association class and **no** full class to describe the relationship!

Many to many - third scenario

- There is **no information** to be kept about this many to many association
  - Such cases are **<u>rare</u>**!
  - Examples: tags, labels, categories, …
- The association is, once again, **<u>unique</u>**
  - There can be only one association between any two specific objects
- In relational DB: requires an "association table"
- Easiest setup, but not always correct!

  ⇒ Use **first scenario** instead!

Many to many - decision tree



Many to many - In Spring

- Scenario 3: least amount of code
  - Actual Spring behavior is more **complicated** than expected!
  - Association table is handled internally by Spring and is handled transparently → "*How do we remove records from the association table?*"
  - Did you know that each bi-directional @ManyToMany has an **owning side**!?
  - "*How does cascading happen? In the other table? In the association table? …*"

```
@ManyToMany
```

Many to many - In Spring

- Scenario 2
  - There's simply an additional class in our domain
  - The additional class can have its own repository
    - **`JournalistRepository`**
    - **`NewspaperRepository`**
    - **`ArticleRepository`** ⇐ It's an entity like any other
  - Clear and predictable implementation
  - Owning side is always clear for '1-*' and '*-1'

`@ManyToOne` and `@OneToMany`

Many to many - In Spring

- Scenario 1
  - Very similar to scenario 2: even if it's modeled as an association class in our domain, we'll still create a regular class in our codebase.
    - Just the **uniqueness** will have to be implemented

```
@Entity
@Table(uniqueConstraints = { @UniqueConstraint(columnNames = { "user_id", "book_id" }) })
public class UserBook {
    @Id
    @GeneratedValue
    private long id;

    @ManyToOne(optional = false, fetch = FetchType.LAZY)
    @JoinColumn(name = "user_id")
    private User user;

    @ManyToOne(optional = false, fetch = FetchType.LAZY)
    @JoinColumn(name = "book_id")
    private Book book;

    // ... add attributes to the relationship
```

Composite Primary Keys in JPA

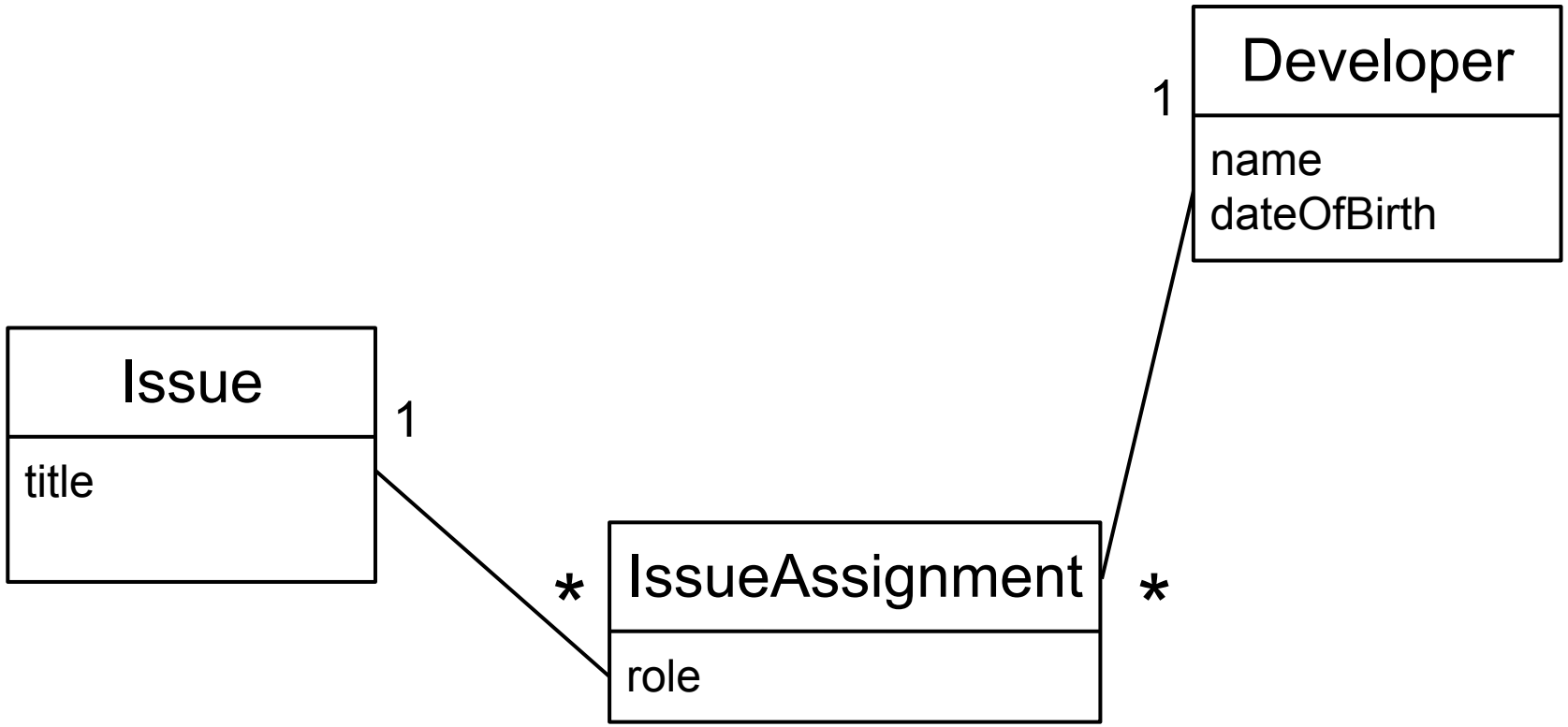
- If the composite key is a **primary** key …
  - Then you'll need a bit more code …
  - You may consider this approach as an **extra**
- Composite Primary Keys in JPA
  - https://www.baeldung.com/jpa-composite-primary-keys

- Example: check the commit on a separate branch of the demo project
  - ACS202: This commit

Many to many - Summary

- **Scenario 1** - Many to many with additional class - unique
  - There is additional data to be stored
  - A name such as ‘BookAuthor’ can be considered okay
  - Use a unique constraint or composite primary key
- **Scenario 2** - Many to many with additional class - not unique
  - There is additional data to be stored
  - Give it a meaningful name - like ‘Article’
- **Scenario 3** - Many to many without additional class - unique *(discouraged)*
  - There is no additional data to be stored

Sample


Developer
name
dateOfBirth
1
Issue
title
1
*
IssueAssignment
role
*