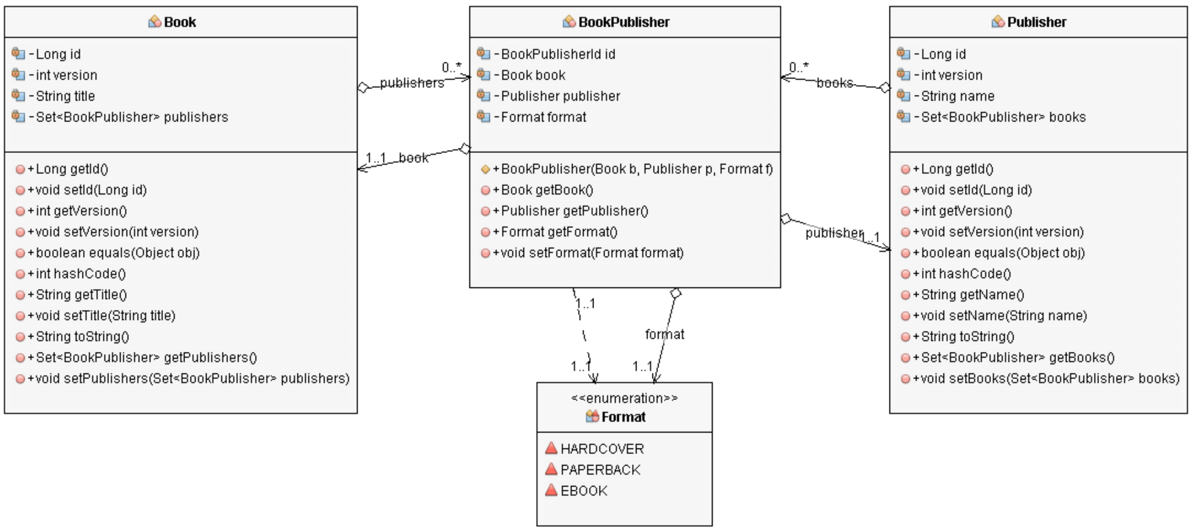
Persist additional attributes for a relationship

1. Model relationship table as an entity



2. Use a @Embeddable class as primary key

Create a class for the primary key and annotate it with *@Embeddable* so that you can use it as a *@EmbeddedId*. Define a mapping for the two foreign key columns.

Make sure to implement the *equals* and *hashCode* methods.

```
@Embeddable
public static class BookPublisherId implements Serializable {

    @Column(name = "fk_book")
    protected Long bookId;

    @Column(name = "fk_publisher")
    protected Long publisherId;
```

3. Define the relationship entity mapping

Use the embeddable class as the primary key.

Map the relationships to the related entities and set *insertable* and *updatable* to *false*.

```
@Entity
public class BookPublisher {

    @EmbeddedId
    private BookPublisherId id;

    @ManyToOne
    @JoinColumn(name = "fk_book",
                insertable = false, updatable = false)
    private Book book;

    @ManyToOne
    @JoinColumn(name = "fk_publisher",
                insertable = false, updatable = false)
    private Publisher publisher;

    @Column
    @Enumerated(EnumType.STRING)
    private Format format;
```

4. Initialize primary key and relationships in constructor

Implement a constructor that initializes all attributes and the relationships.

```
@Entity
public class BookPublisher {

        public BookPublisher(Book b, Publisher p, Format f) {
                // create primary key
                this.id = new BookPublisherId(b.getId(), p.getId());

                // initialize attributes
                this.book = b;
                this.publisher = p;
                this.format = f;

                // update relationships to assure referential integrity
                p.getBooks().add(this);
                b.getPublishers().add(this);
        }
```