Fluent Entities with Hibernate and JPA

Fluent interfaces are a popular API design pattern in the Java world. The goal of the pattern is to create APIs that are very easy to read, and that define something similar to a domain-specific language. To achieve that, the API heavily relies on method chaining so that the code that uses the API flows and almost reads like prose.

A basic example of a fluent API might look like this:

```
Book b = new Book()
    .title("Hibernate Tips - More than 70 solutions to common Hibernate problems")
    .publishedOn(LocalDate.of(2017, 04, 01))
    .writtenBy(authors);
```

APIs like these are often used for value classes and configuration data. So, it's no surprise that a lot of teams would like to use them for entities.

Unfortunately, that's not as easy as you might expect.

JavaBeans Convention vs. Fluent Interfaces

The JPA specification requires your entities to follow the JavaBeans Introspector convention.

"In this case, for every persistent property property of type T of the entity, there is a getter method, getProperty, and setter method setProperty. For boolean properties, isProperty may be used as an alternative name for the getter method.[2]

For single-valued persistent properties, these method signatures are:

- *T getProperty()*
- *void setProperty(T t)"*

JSR 338: *JavaTM Persistence API, Version 2.2 – Section 2.2 (p. 24)*

So, if you want to create JPA-compliant entities, you need to implement public getter and setter methods for all entity attributes. If you also wish to provide a fluent interface API, you can only add these methods so that both APIs coexist on the same class.

Hibernate is not as strict as the JPA specification, but it still recommends to follow the JavaBeans convention.

"Although Hibernate does not require it, it is recommended to follow the JavaBean conventions and define getters and setters for entity persistent attributes."

Hibernate User Guide – Section 2.5.4

The interesting part in this quote is "Hibernate does not require it". So, if you decide to ignore the JPA specification and potential portability issues, Hibernate allows you to implement a clean, fluent interface API for your entities.

JPA-Compliant Implementation

If you want to implement your entities in a JPA-compliant way, you will not be able to design a nice and clean fluent interface API. You can only add the fluent methods to your entity class, and keep the getter and setter methods.

```
@Entity
public class Book {

   @Id
   @GeneratedValue
   private Long id;

   @Version
   private int version;

   private String title;

   private LocalDate publishingDate;

   @ManyToMany
   private Set<Author> authors = new HashSet<Author>();

   public Long getId() {
      return id;
   }

   public int getVersion() {
      return version;
   }

   public String getTitle() {
      return title;
   }

   public void setTitle(String title) {
      this.title = title;
   }

   public Book title(String title) {
      this.title = title;
      return this;
   }

   public Set<Author> getAuthors() {
      return authors;
   }

   public void setAuthors(Set<Author> authors) {
      this.authors = authors;
   }

   public Book writtenBy(Set<Author> authors) {
      this.authors = authors;
      return this;
   }
public LocalDate getPublishingDate() {
      return publishingDate;
   }

   public void setPublishingDate(LocalDate publishingDate) {
      this.publishingDate = publishingDate;
   }

   public Book publishedOn(LocalDate publishingDate) {
      this.publishingDate = publishingDate;
      return this;
   }
}
```

When you use the Book entity in your business code, you can decide which API you want to use.

You can call the getter and setter methods.

```
Book b = new Book();
b.setTitle("Hibernate Tips - More than 70 solutions to common Hibernate
problems");
b.setPublishingDate(LocalDate.of(2017, 04, 01));
b.setAuthors(authors);
em.persist(b);
```

Or you can use the methods of the fluent interface API.

```
Book b = new Book()
            .title("Hibernate Tips - More than 70 solutions to common Hibernate
problems")
            .publishedOn(LocalDate.of(2017, 04, 01))
            .writtenBy(authors);
em.persist(b);
```

Hibernate-Compliant Implementation

As described earlier, Hibernate recommends providing getter and setter methods that follow the JavaBeans convention. But it doesn't require these methods.

That gives you higher flexibility when designing the API of your entities. You can decide if you provide a fluent API by changing the setter methods so that they return the entity object or if you provide different methods instead.

Let's take a look at both options.

Fluent Entities with Setter Methods

Changing the return type of your setter methods is a simple and very flexible approach.

```
@Entity
public class Book {

   @Id
   @GeneratedValue
   private Long id;

   @Version
   private int version;

   private String title;

   private LocalDate publishingDate;

   @ManyToMany
   private Set<Author> authors = new HashSet<Author>();

   public Long getId() {
      return id;
   }
public int getVersion() {
      return version;
   }

   public String getTitle() {
      return title;
   }
```

```
    public Book setTitle(String title) {
        this.title = title;
        return this;
    }

    public Set<Author> getAuthors() {
        return authors;
    }

    public Book setAuthors(Set<Author> authors) {
        this.authors = authors;
        return this;
    }

    public LocalDate getPublishingDate() {
        return publishingDate;
    }

    public Book setPublishingDate(LocalDate publishingDate) {
        this.publishingDate = publishingDate;
        return this;
    }
}
```

The API of your entity looks almost identical to a JPA-compliant entity. And you can still call all setter methods as you would usually do. That enables you to add the fluent API without changing any existing business code.

```
Book b = new Book()
        .title("Hibernate Tips - More than 70 solutions to common Hibernate problems")
        .publishedOn(LocalDate.of(2017, 04, 01))
        .writtenBy(authors);
em.persist(b);
```

And you can also chain the calls of your setter methods to use them fluently.

```
Book b = new Book()
          .setTitle("Hibernate Tips - More than 70 solutions to common Hibernate
problems")
          .setPublishingDate(LocalDate.of(2017, 04, 01))
          .setAuthors(authors);
em.persist(b);
```

But this code snippet also shows the downside of this approach. Even though you chain the method calls, the code still reads like you're calling multiple setter methods. It doesn't flow.

Fluent Entities without Setter Methods

If you want to take it one step further, you need to annotate your primary key attribute with @Id. That tells Hibernate to use the field-based access strategy.

You then don't need to provide any setter or getter methods for your entity attributes. That enables you to rename the setter methods to create a real fluent interface.

```
@Entity
public class Book {

   @Id
   @GeneratedValue
   private Long id;

   @Version
   private int version;

   private String title;

   private LocalDate publishingDate;
```

```
    @ManyToMany
    private Set<Author> authors = new HashSet<Author>();

    public Long getId() {
        return id;
    }

    public int getVersion() {
        return version;
    }

    public String getTitle() {
        return title;
    }

    public Book title(String title) {
        this.title = title;
        return this;
    }

    public Set<Author> getAuthors() {
        return authors;
    }

    public Book writtenBy(Set<Author> authors) {
        this.authors = authors;
        return this;
    }

    public LocalDate getPublishingDate() {
        return publishingDate;
    }

    public Book publishedOn(LocalDate publishingDate) {
        this.publishingDate = publishingDate;
        return this;
    }
}
```

As you can see in the following code snippet, the new method names drastically improve the readability of the business code. It now almost reads like you're describing the Book and it's no longer a list of technical calls of setter methods.

```
Book b = new Book()
   .title("Hibernate Tips - More than 70 solutions to common Hibernate problems")
   .publishedOn(LocalDate.of(2017, 04, 01))
   .writtenBy(authors);
em.persist(b);
```

One thing you need to be aware of when you implement your API like this is that the fluent interface API doesn't affect your JPQL queries. You still use the attribute names in your query. That might get a little bit confusing because they no longer match the method names that you use in your business code.

```
TypedQuery<Book> q = em.createQuery("SELECT b FROM Book b WHERE
b.publishingDate = :publishingDate", Book.class);
q.setParameter("publishingDate", LocalDate.of(2017, 04, 01));
List<Book> books = q.getResultList();
```