Introduction to Hibernate

Architecture. Domain Model. Bootstrapping

About me

Trayan Iliev

– CEO of **IPT – Intellectual Products & Technologies**
http://www.iproduct.org

– Oracle® certified programmer 15+ Y

– end-to-end reactive fullstack apps with Java, ES6+, TypeScript, Angular, React and Vue.js

– 12+ years IT trainer: Spring, Java EE, Node.js, Express, GraphQL, SOA, REST, DDD & Reactive Microservices

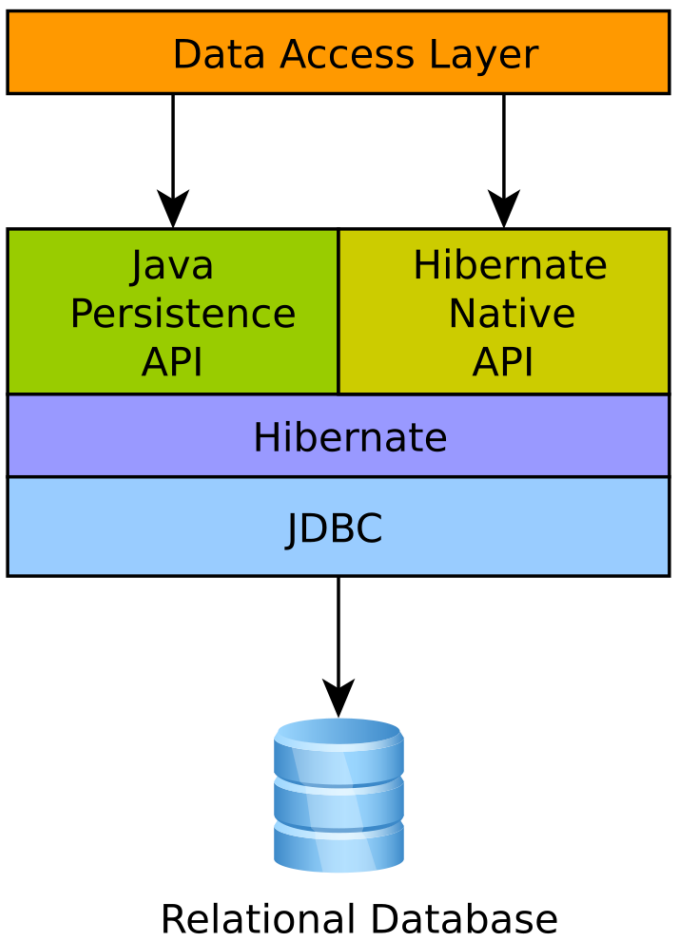
– Voxxed Days, jPrime, Java2Days, jProfessionals, BGOUG, BGJUG, DEV.BG speaker

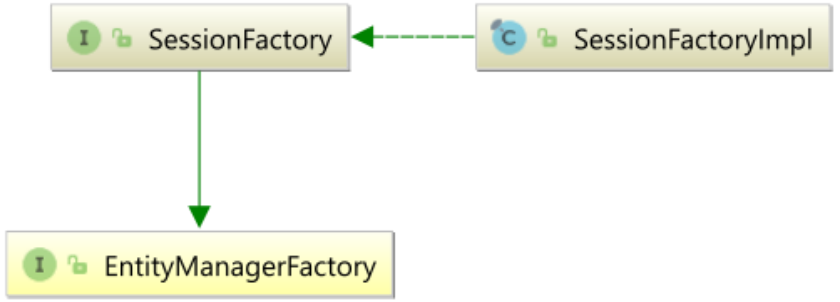
– Organizer RoboLearn hackathons and IoT enthusiast

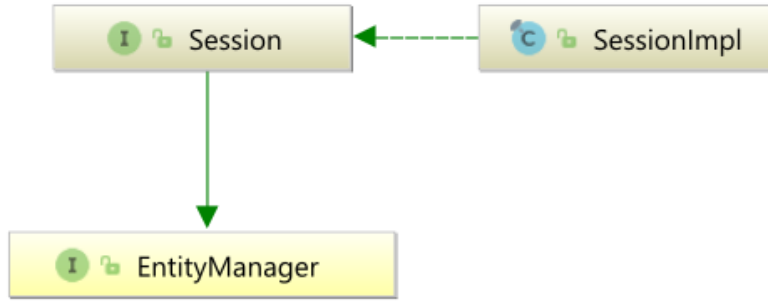
Where to Find The Code and Materials?

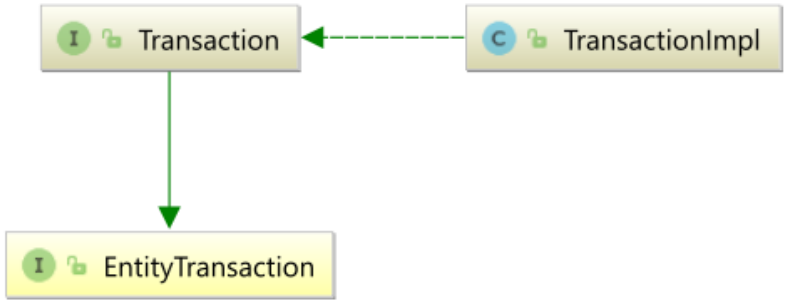
https://github.com/iproduct/course-hibernate

Hibernate Architecture









Hibernate Architecture - II

- **SessionFactory (org.hibernate.SessionFactory)** - a thread-safe (and immutable) representation of the mapping of the application domain model to a database. Acts as a factory for org.hibernate.Session instances. The EntityManagerFactory is the JPA equivalent of a SessionFactory and basically, those two converge into the same SessionFactory implementation.
- Very expensive to create, so, for any given database, the application should have only one associated SessionFactory.
- The SessionFactory maintains services that Hibernate uses across all Session(s) such as second level caches, connection pools, transaction system integrations, etc.

Hibernate Architecture - III

- **Session (org.hibernate.Session)** - a single-threaded, short-lived object conceptually modeling a "Unit of Work" (PoEAA). In JPA nomenclature, the Session is represented by an EntityManager.
- Behind the scenes, the Hibernate Session wraps a JDBC java.sql.Connection and acts as a factory for org.hibernate.Transaction instances. It maintains a generally "repeatable read" persistence context (first level cache) of the application domain model.
- **Transaction (org.hibernate.Transaction)** - a single-threaded, short-lived object used by the application to demarcate individual physical transaction boundaries. EntityTransaction is the JPA equivalent and both act as an abstraction API to isolate the application from the underlying transaction system in use (JDBC or JTA).

Domain Model

- Historically applications using Hibernate would have used its proprietary XML mapping file format for this purpose. With the coming of JPA, most of this information is now defined in a way that is portable across ORM/JPA providers using annotations (and/or standardized XML format).
- We usually prefer the JPA mappings where possible.
- For Hibernate mapping features not supported by JPA we will prefer Hibernate extension annotations.

Mapping Types

- Hibernate understands both the Java and JDBC representations of application data.
- Hibernate type – provides the ability to read/write this data from/to the database. It is an implementation of the org.hibernate.type.Type interface. Also describes various behavioral aspects of the Java type such as how to check for equality, how to clone values, etc.
- Hibernate type is **neither a Java type nor a SQL data type**. It provides information about mapping a Java type to an SQL type as well as how to persist and fetch a given Java type to and from a relational database.
- When you encounter the term type in discussions of Hibernate, it may refer to the Java type, the JDBC type, or the Hibernate type, depending on the context.

Hiberbate Native Bootstrapping

- There are two types of ServiceRegistry and they are hierarchical:
- **BootstrapServiceRegistry**, which has no parent and holds these three required services:
  - ClassLoaderService: allows Hibernate to interact with the ClassLoader of the various runtime environments
  - IntegratorService: controls the discovery and management of the Integrator service allowing third-party applications to integrate with Hibernate
  - StrategySelector: resolves implementations of various strategy contracts
- **StandardServiceRegistry**

Building BootstrapServiceRegistry

```
BootstrapServiceRegistry bootstrapServiceRegistry =
        new BootstrapServiceRegistryBuilder()
                .applyClassLoader()
                .applyIntegrator()
                .applyStrategySelector()
                .build();
```

Building StandardServiceRegistry

```
BootstrapServiceRegistry bootstrapServiceRegistry =
        new BootstrapServiceRegistryBuilder().build();
StandardServiceRegistryBuilder standardServiceRegistryBuilder =
        new StandardServiceRegistryBuilder(bootstrapServiceRegistry);
StandardServiceRegistry standardServiceRegistry = standardServiceRegistryBuilder
        .configure()
        .build();
```

Building Metadata

```
MetadataSources metadataSources =
        new MetadataSources(standardServiceRegistry);
metadataSources.addPackage( … );
metadataSources.addAnnotatedClass( … );
metadataSources.addResource( … )
Metadata metadata = metadataSources.buildMetadata();
```

Building and Using SessionFactory and Session

```
// Get SessionFactory
SessionFactory sessionFactory = metadata.getSessionFactoryBuilder().build();

// Get Session
Session session = sessionFactory.openSession();

// Persist entity
Contact contact = new Contact(1,
        new Name("Ivan", "Dimitrov", "Petrov"),
        "From work", new URL("http://ivan.petrov.me/"), true);
session.beginTransaction();
session.persist(contact);
session.getTransaction().commit();
session.close();
sessionFactory.close();
```

WEB-INF/applicationContext.xml -I

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx.xsd
        http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/aop/spring-aop.xsd">

    <context:property-placeholder location="classpath:jdbc.properties" />

    <context:component-scan base-package="org.iproduct.spring.webmvc.dao,
         org.iproduct.spring.webmvc.service"/>

    <context:annotation-config />
```

Source: https://docs.spring.io/spring-framework/docs/

WEB-INF/applicationContext.xml -II

```
<bean id="dataSource" class="org.apache.commons.dbcp2.BasicDataSource"
        destroy-method="close">
    <property name="driverClassName" value="${jdbc.driverClassName}" />
    <property name="url" value="${jdbc.url}" />
    <property name="username" value="${jdbc.username}" />
    <property name="password" value="${jdbc.password}" />
</bean>

<bean id="sessionFactory"
      class="org.springframework.orm.hibernate5.LocalSessionFactoryBean">
    <property name="dataSource" ref="dataSource"/>
    <property name="mappingResources">
        <list><value>article.hbm.xml</value></list>
    </property>
    <property name="hibernateProperties">
        <value>
            hibernate.dialect=org.hibernate.dialect.HSQLDialect
            hibernate.hbm2ddl.auto=update
        </value>
    </property>
</bean>
```

Source: https://docs.spring.io/spring-framework/docs/

WEB-INF/applicationContext.xml III

```
   <bean id="transactionManager"
class="org.springframework.orm.hibernate5.HibernateTransactionManager">
       <property name="sessionFactory" ref="sessionFactory"/>
    </bean>

    <tx:annotation-driven/>

</beans>
```

Source: https://docs.spring.io/spring-framework/docs/

Hibernate Mapping: article.hbm.xml

```
<hibernate-mapping>
    <class name="org.iproduct.spring.webmvc.model.Article"
table="ARTICLES">

        <meta attribute="class-description">
            This class contains the articles details.
        </meta>

        <id name="id" type="long" column="id">
            <generator class="identity"/>
        </id>

        <property name="title" column="title" type="string"/>
        <property name="content" column="content" type="string"/>
        <property name="createdDate" column="created_date"
type="timestamp"/>
        <property name="pictureUrl" column="picture_url" type="string"/>

    </class>
</hibernate-mapping>
```

Source: https://docs.spring.io/spring-framework/docs/

ArticlesDaoHibernate Class - I

```
@Repository
@Transactional
public class ArticleDaoHibernate implements ArticleDao {

    private SessionFactory sessionFactory;

    @Autowired
    public void setSessionFactory(SessionFactory sessionFactory) {
        this.sessionFactory = sessionFactory;
    }

    @Override
    public Collection<Article> findAll() {
        return this.sessionFactory.getCurrentSession()
          .createQuery("select article from Article article",
Article.class)
          .list();
    }
    @Override
    public Article find(long id) {
        return this.sessionFactory.getCurrentSession()
          .byId(Article.class).load(id);
    }
```

Source: https://docs.spring.io/spring-framework/docs/

ArticlesDaoHibernate Class - II

```
@Override
 public Article create(Article article) {
     this.sessionFactory.getCurrentSession()
             .persist(article);
     return article;
 }
 @Override
 public Article update(Article article) {
     Article toBeDeleted = find(article.getId());
     if (toBeDeleted == null) {
         throw new EntityNotExistException("Article "+article.getId()+" not exist.");
     }
     return (Article) this.sessionFactory.getCurrentSession()
         .merge(article);
 }
 @Override
 public Article remove(long articleId) {
     Article toBeDeleted = find(articleId);
     if (toBeDeleted == null) {
        throw new EntityNotExistException("Article "+article.getId()+" not exist.");
     }
     this.sessionFactory.getCurrentSession()
         .delete(toBeDeleted);
     return toBeDeleted;
 }}
```

Source: https://docs.spring.io/spring-framework/docs/

Java Persistence API (JPA)

- **JPA four main parts:**
  - Java Persistence API
  - JPA Query Language
  - Java Persistence Criteria API
  - Object to Relational Mapping (ORM) metadata
- **JPA Entity Classes**
  - persistent fields
  - persistent properties
- **@Entity** annotation

Object-Relational Mapping (ORM)

- Package: javax.persistence
- Simple keys - @Id annotation
- Composite keys
  - Primary Key Class – requirements and structure
  - Annotations – @EmbeddedId, @IdClass
- Realtions between entity objects –
  - uni- and bi-directional,
  - 1:1, 1:many, many:1 many:many

Advantages of Spring ORM

- Easier testing
- Common data access exceptions
- General resource management
- Integrated transaction management

ORM Cascade Updates

- Entities that have a dependency relationship can be managed declaratively by JPA using CascadeType:

– **ALL** – всички операции са каскадни

– **DETACH** – каскадно отстраняване

– **MERGE** – каскадно сливане

– **PERSIST** – каскадно персистиране

– **REFRESH** – каскадно обновяване

– **REMOVE** – каскадно премахване

- **@OneToMany(cascade=REMOVE, mappedBy="customer")**
- **public Set<Order> getOrders() { return orders; }**

Entity Embeddables

- **@Embeddable** – анотира клас, който не е Entity, но може да бъде част от Entity
- **@Embedded** – embeds Embeddable class into Entity class
- Embedding can be hierarchical on multiple levels
- Annotations: **@AttributeOverride, @AttributeOverrides, @AssociationOverride, @AssociationOverrides**

Entity Inheritance

- Entity / Abstract entity
- Mapped superclass
- Non-entity superclass
- Entity -> DB tables mapping strategies
  - SingleTable per Class Hierarchy
  - TheTable per Concrete Class
  - The Joined Subclass Strategy

Persistent Units

- Persistent Unit description in persistence.xml file:
  - description
  - provider
  - jta-data-source
  - non-jta-data-source
  - mapping-file
  - jar-file
  - class
  - exclude-unlisted-classes
  - properties

Persistent Unit Example 1

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
version="1.0"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd">
    <persistence-unit name="CustomerDBPU" transaction-type="JTA">
        <jta-data-source>jdbc/sample</jta-data-source>
        <class>customerdb.Customer</class>
        <class>customerdb.DiscountCode</class>
        <properties/>
    </persistence-unit>
</persistence>
```

Persistent Unit Example 2 - I

- <?xml version="1.0" encoding="UTF-8"?>
- <persistence version="1.0"
  xmlns="http://java.sun.com/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
  http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd">
- <persistence-unit name="invoicingPU"
- transaction-type="RESOURCE_LOCAL">
- <provider>oracle.toplink.essentials.PersistenceProvider</provider>
- <class>myinvoice.dbentities.ProductDB</class>
- <class>myinvoice.dbentities.PositionDB</class>
- <class>myinvoice.dbentities.InvoiceDB</class>
-

Persistent Unit Example 2 - II

```
        <class>myinvoice.dbentities.ContragentDB</class>
        <properties>
            <property name="toplink.jdbc.user" value="root"/>
            <property name="toplink.jdbc.password"
value="root"/>
            <property name="toplink.jdbc.url"

value="jdbc:mysql://localhost:3306/invoicing"/>
            <property name="toplink.jdbc.driver"
                value="com.mysql.jdbc.Driver"/>
        </properties>
    </persistence-unit>
</persistence>
```

Collection Type Persistent Fields

- Field or properties should be of Collection or Map type (usually generic):
  - java.util.Collection
  - java.util.Set
  - java.util.List
  - java.util.Map
- @ElementCollection
- @CollectionTable – name of additional table
- @Embeddable, @Column
- @AttributeOverride, @AttributeOverrides

Main JPA Annotations

- @PersistenceUnit,
- @PersistenceContext
- @Entity
- @Id
- @OneToOne
- @OneToMany
- @ManyToMany
- @DiscrimainatorColumn
- @Column
- @JoinTable
- @JoinColumn
- @Embeddable
- @Embedded

JPA Entity Annotations Example

```
@Entity
public class Article {
    @Id
    @GeneratedValue
    private Long id;

    @Length(min=3, max=80)
    private String title;

    @Length(min=3, max=2048)
    private String content;

    @NotNull
    @ManyToOne
    @JoinColumn(name="AUTHOR_ID",
nullable=false)
    private User author;

    @Length(min=3, max=256)
    private String pictureUrl;

    @Temporal(TemporalType.TIMESTAMP)
    private Date created = new Date();

    @Temporal(TemporalType.TIMESTAMP)
    private Date updated = new Date();
… }
```

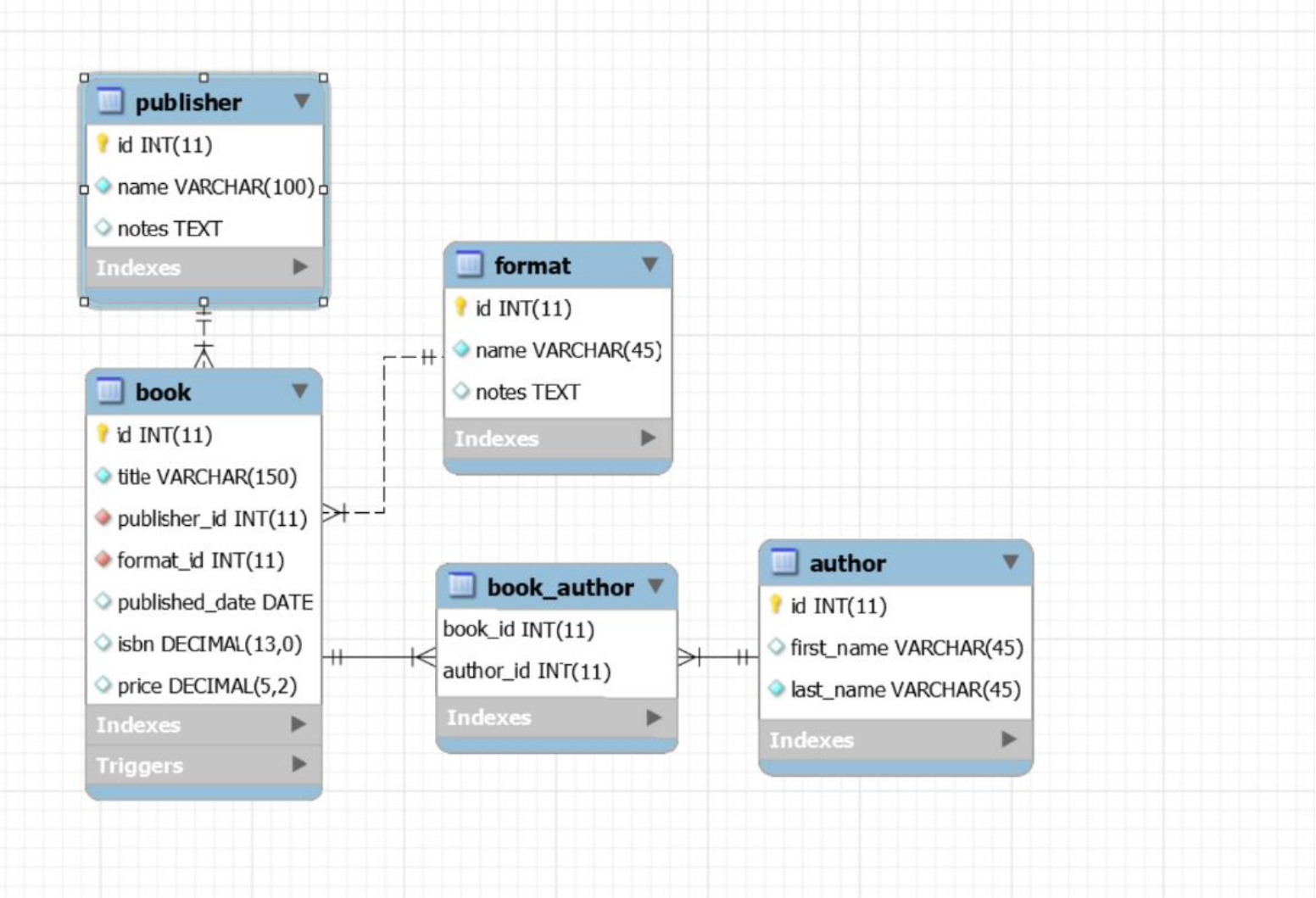
```
@Entity
public class User implements UserDetails {
    @Id
    @GeneratedValue
    private long id;

    @NotNull
    @Length(min = 3, max = 30)
    private String username;
    ...

    @NonNull
    private String roles = "ROLE_USER";

    @OneToMany(mappedBy = "author",
                cascade = CascadeType.ALL,
                orphanRemoval=true)
    Collection<Article> articles =
        new ArrayList<>();

    @Temporal(TemporalType.TIMESTAMP)
    private Date created = new Date();

    @Temporal(TemporalType.TIMESTAMP)
    private Date updated = new Date();
… }
```

JPA Entities: @ManyToMany

```
@Entity
public class Book {
    @Id @GeneratedValue
    private int id;

    @NotNull
    private String title;

    @ManyToOne
    @JoinColumn(name = "PUBLISHER_ID",
                referencedColumnName = "id")
    private Publisher publisher;

    @Column(name = "PUBLISHED_DATE") @PastOrPresent
    @DateTimeFormat(iso = DateTimeFormat.ISO.DATE)
    private LocalDate publishedDate;

    @Pattern(regexp = "\\d{10}|\\d{13}")
    private String isbn;

    @NotNull @Min(0)
    private double price;

    @ManyToMany(fetch = FetchType.EAGER)
    @JoinTable(name="BOOK_AUTHOR", joinColumns=
@JoinColumn(name="BOOK_ID",referencedColumnName="ID"),
                inverseJoinColumns=
@JoinColumn(name="AUTHOR_ID",referencedColumnName="ID")
    )
    private List<Author> authors = new ArrayList<>();
… }
```

```
@Entity
public class Author {
        @Id @GeneratedValue
        private int id;

        @NotNull
        @Length(min=2, max=60)
        @Column(name = "first_name")
        private String firstName;

        @NotNull
        @Length(min=2, max=60)
        @Column(name = "last_name")
        private String lastName;

        @ManyToMany(mappedBy = "authors",
                    fetch =
FetchType.EAGER)
        List<Book> books = new
ArrayList<>();
… }
```

JPA Entities: ER Diagram



Java Persistence Query Language

- Object-oriented database queries
- Navigation
- Abstract schema
- Path expression
- State field
- Relationship field

Java Persistence Query Language

- SELECT
- FROM
- WHERE
- GROUP BY
- HAVING
- ORDER BY
- UPDATE
- DELETE
- AS, IN
- LIKE
- EXISTS, ANY, ALL
- NEW

JPA Setup in Spring

- ```
  <beans>
      <bean id="myEmf"
  class="org.springframework.orm.jpa.LocalEntityManagerFactoryBean">
          <property name="persistenceUnitName" value="myPersistenceUnit"/>
      </bean>
  </beans>
  ```

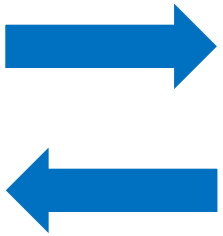
- ```
  <beans>
      <jee:jndi-lookup id="myEmf" jndi-name="persistence/myPersistenceUnit"/>
  </beans>
  ```

- ```
  <beans>
      <bean id="myEmf"
  ```
- ```
  class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
          <property name="dataSource" ref="someDataSource"/>
          <property name="loadTimeWeaver">
              <bean
  class="org.springframework.instrument.classloading.InstrumentationLoadTimeWeaver"
  />
          </property>
      </bean>
  </beans>
  ```

Mapping Example

```java
@Entity(name = "Contact")
@Data
public class Contact {

    @Id
    private Integer id;

    @Embedded
    private Name name;

    private String notes;

    private URL website;

    private boolean starred;

    public Name getName() {
        return name;
    }
}
```

```java
@Embeddable
@Data
public class Name {

    private String firstName;

    private String middleName;

    private String lastName;
}
```



```
create table Contact
(
    id      integer not null,
    first   varchar(255),
    last    varchar(255),
    middle  varchar(255),
    notes   varchar(255),
    starred boolean not null,
    website varchar(255),
    primary key (id)
)
```

Code First

DB Schema First

Value and Entity Types

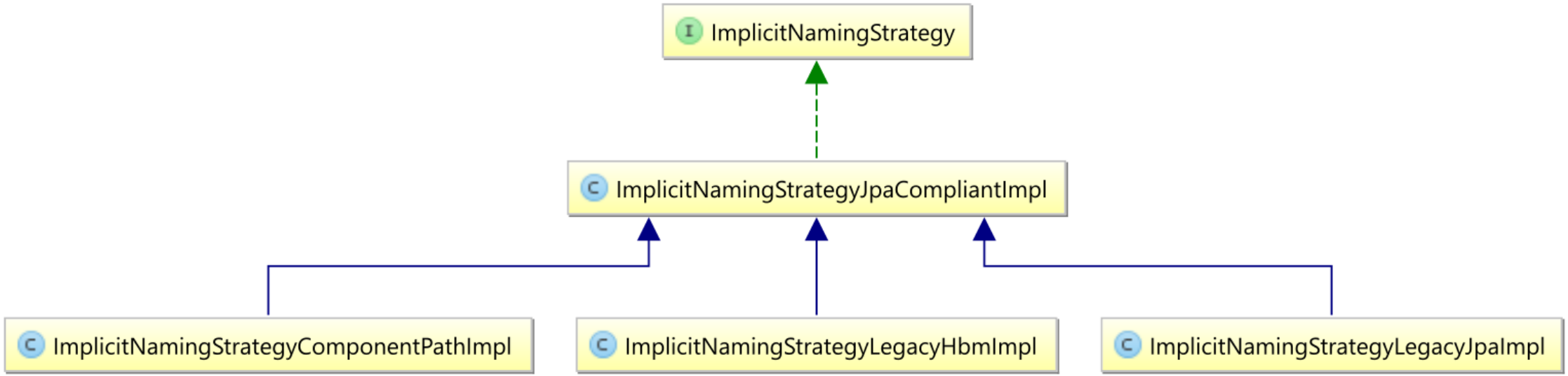
- **Value types** - a value type is a piece of data that does not define its own lifecycle. It is, in effect, owned by an entity, which defines its lifecycle -> persistent attributes.
  - Basic types – e.g. in mapping the Contact table, all attributes except for name would be basic types.
  - Embeddable types - the name attribute is an example of an embeddable type, which is discussed in details in Embeddable types
  - Collection types - although not featured in the aforementioned example, collection types are also a distinct category among value types. Collection types are
- **Entity types** - by nature of their unique identifier, entities exist independently and define their own lifecycle, whereas values do not. Entities are domain model classes which correlate to rows in a database table, using a unique identifier.

Naming Strategies

Part of the mapping of an object model to the relational database is mapping names from the object model to the corresponding database names. Hibernate looks at this as 2-stage process:

1. The first stage is determining a proper logical name from the domain model mapping. A logical name can be either explicitly specified by the user (e.g., using @Column or @Table) or it can be implicitly determined by Hibernate through an ImplicitNamingStrategy contract.
2. Second is the resolving of this logical name to a physical name which is defined by the PhysicalNamingStrategy contract.

Naming Strategies



Implicite Naming Strategies

- Hibernate defines multiple ImplicitNamingStrategy implementations out-of-the-box. Applications are also free to plug in custom implementations.
- Applications can specify the implementation using the hibernate.implicit_naming_strategy configuration setting which accepts:
  - **default** – for org.hibernate.boot.model.naming.ImplicitNamingStrategyJpaCompliantImpl - an alias for jpa
  - **jpa** - for org.hibernate.boot.model.naming.ImplicitNamingStrategyJpaCompliantImpl - the JPA 2.0 compliant naming strategy
  - **legacy-hbm** - for org.hibernate.boot.model.naming.ImplicitNamingStrategyLegacyHbmImpl - compliant with the original Hibernate NamingStrategy
  - **legacy-jpa** - for org.hibernate.boot.model.naming.ImplicitNamingStrategyLegacyJpaImpl - compliant with the legacy NamingStrategy developed for JPA 1.0, which was unfortunately unclear in many respects regarding implicit naming rules
  - **component-path** –org.hibernate.boot.model.naming.ImplicitNamingStrategyComponentPathImpl - mostly follows ImplicitNamingStrategyJpaCompliantImpl rules, except that it uses the full composite paths, as opposed to just the ending property part – e.g.
- By calling org.hibernate.boot.MetadataBuilder#applyImplicitNamingStrategy

Physical Naming Strategies

- There are multiple ways to specify the PhysicalNamingStrategy to use. First, applications can specify the implementation using the hibernate.physical_naming_strategy configuration setting which accepts:
  - reference to a Class that implements the org.hibernate.boot.model.naming.PhysicalNamingStrategy contract
  - FQN of a class that implements the org.hibernate.boot.model.naming.PhysicalNamingStrategy contract
- Secondly, applications and integrations can leverage org.hibernate.boot.MetadataBuilder#applyPhysicalNamingStrategy

Basic Types

- Basic value types usually map a single database column, to a single, non-aggregated Java type
- Internally Hibernate uses a registry of basic types when it needs to resolve a specific org.hibernate.type.Type:

https://docs.jboss.org/hibernate/orm/5.6/userguide/html_single/Hibernate_User_Guide.html#basic-provided

- The @Basic annotation - defines 2 attributes:
  - optional - boolean (defaults to true) - whether this attribute allows nulls.
  - fetch - FetchType (defaults to EAGER) - whether should be fetched eagerly or lazily. Hibernate ignores this setting for basic types unless you are using bytecode enhancement.

Explicite Basic Types

```
@Entity(name = "Product")
 public class Product {
        @Id
        private Integer id; private String sku;

        @org.hibernate.annotations.Type( type = "nstring" )
        private String name;

        @org.hibernate.annotations.Type( type = "materialized_nclob" )
        private String description;
}
```

Basic Types

- Basic value types usually map a single database column, to a single, non-aggregated Java type
- Internally Hibernate uses a registry of basic types when it needs to resolve a specific org.hibernate.type.Type:

https://docs.jboss.org/hibernate/orm/5.6/userguide/html_single/Hibernate_User_Guide.html#basic-provided

- The @Basic annotation - defines 2 attributes:
  - optional - boolean (defaults to true) - whether this attribute allows nulls.
  - fetch - FetchType (defaults to EAGER) - whether should be fetched eagerly or lazily. Hibernate ignores this setting for basic types unless you are using bytecode enhancement.

Embeddable Types

```java
@Embeddable
public class Publisher {

    private String name;


    @Embedded
    private Location location;

    public Publisher(String name, Location location) {
        this.name = name;
        this.location = location;
    }

    private Publisher() {}

    //Getters and setters are omitted for brevity
}
```

```java
@Embeddable
public class Location {

    private String country;

    private String city;

    public Location(String country, String city) {
        this.country = country;
        this.city = city;
    }

    private Location() {}

    //Getters and setters are omitted for brevity
}
```

Embeddable Types - II

```java
@Entity(name = "Book")
public class Book {

    @Id
    @GeneratedValue
    private Long id;

    private String title;

    private String author;

    @Embedded
    private Publisher2 publisher;

    public Publisher2 getPublisher() {
        return publisher;
    }
}
```

```java
@Embeddable
@Data
class Publisher2 {

    @Column(name = "publisher_name")
    private String name;

    @Column(name = "publisher_country")
    private String country;
}
```

Embeddable Types – Attribute Overrides

```
@Entity(name = "Book")
@AttributeOverrides({
        @AttributeOverride(
                name = "ebookPublisher.name",
                column = @Column(name = "ebook_publisher_name")
        ),
        @AttributeOverride(
                name = "paperBackPublisher.name",
                column = @Column(name = "paper_back_publisher_name")
        )
})
@AssociationOverrides({
        @AssociationOverride(
                name = "ebookPublisher.country",
                joinColumns = @JoinColumn(name = "ebook_publisher_country_id")
        ),
        @AssociationOverride(
                name = "paperBackPublisher.country",
                joinColumns = @JoinColumn(name =
"paper_back_publisher_country_id")
        )
})
```

```
public class Book2 {
    @Id
    @GeneratedValue
    private Long id;

    private String title;

    private String author;

    @Embedded
    private Publisher2 ebookPublisher;

    @Embedded
    private Publisher2 paperBackPublisher;

    public Publisher2 getPaperBackPublisher() {
        return paperBackPublisher;
    }

    public Publisher2 getEbookPublisher() {
        return ebookPublisher;
    }
}
```

Embeddable Types – Attribute Overrides

```
@Entity(name = "Book")
@AttributeOverrides({
        @AttributeOverride(
                name = "ebookPublisher.name",
                column = @Column(name = "ebook_publisher_name")
        ),
        @AttributeOverride(
                name = "paperBackPublisher.name",
                column = @Column(name = "paper_back_publisher_name")
        )
})
@AssociationOverrides({
        @AssociationOverride(
                name = "ebookPublisher.country",
                joinColumns = @JoinColumn(name = "ebook_publisher_country_id")
        ),
        @AssociationOverride(
                name = "paperBackPublisher.country",
                joinColumns = @JoinColumn(name =
"paper_back_publisher_country_id")
        )
})
```

```
public class Book2 {
    @Id
    @GeneratedValue
    private Long id;

    private String title;

    private String author;

    @Embedded
    private Publisher2 ebookPublisher;

    @Embedded
    private Publisher2 paperBackPublisher;

    public Publisher2 getPaperBackPublisher() {
        return paperBackPublisher;
    }

    public Publisher2 getEbookPublisher() {
        return ebookPublisher;
    }
}
```

@Target Mapping

```
@Embeddable
class GPS implements Coordinates {
    private double latitude;
    private double longitude;

    public GPS() {
    }

    public GPS(double latitude, double longitude) {
        this.latitude = latitude;
        this.longitude = longitude;
    }
    @Override
    public double x() {
        return latitude;
    }
    @Override
    public double y() {
        return longitude;
    }
}
```

```
interface Coordinates {
    double x();
    double y();
}
```

```
@Entity(name = "City")
public class City {

    @Id
    @GeneratedValue
    private Long id;

    private String name;

    @Embedded
    @Target( GPS.class )
    private Coordinates coordinates;
}
```

@Target Mapping

```java
@Embeddable
class GPS implements Coordinates {
    private double latitude;
    private double longitude;

    public GPS() {
    }

    public GPS(double latitude, double longitude) {
        this.latitude = latitude;
        this.longitude = longitude;
    }
    @Override
    public double x() {
        return latitude;
    }
    @Override
    public double y() {
        return longitude;
    }
}
```

```java
interface Coordinates {
    double x();
    double y();
}
```

```java
@Entity(name = "City")
public class City {

    @Id
    @GeneratedValue
    private Long id;

    private String name;

    @Embedded
    @Target( GPS.class )
    private Coordinates coordinates;
}
```

References

- [PoEAA] Martin Fowler. Patterns of Enterprise Application Architecture. Addison-Wesley Professional. 2002.
- [JPwH] Christian Bauer & Gavin King. Java Persistence with Hibernate, Second Edition. Manning. 2015.

Thank's for Your Attention!

Trayan Iliev

IPT – Intellectual Products & Technologies

http://iproduct.org/

https://github.com/iproduct

https://twitter.com/trayaniliev

https://www.facebook.com/IPT.EACAD