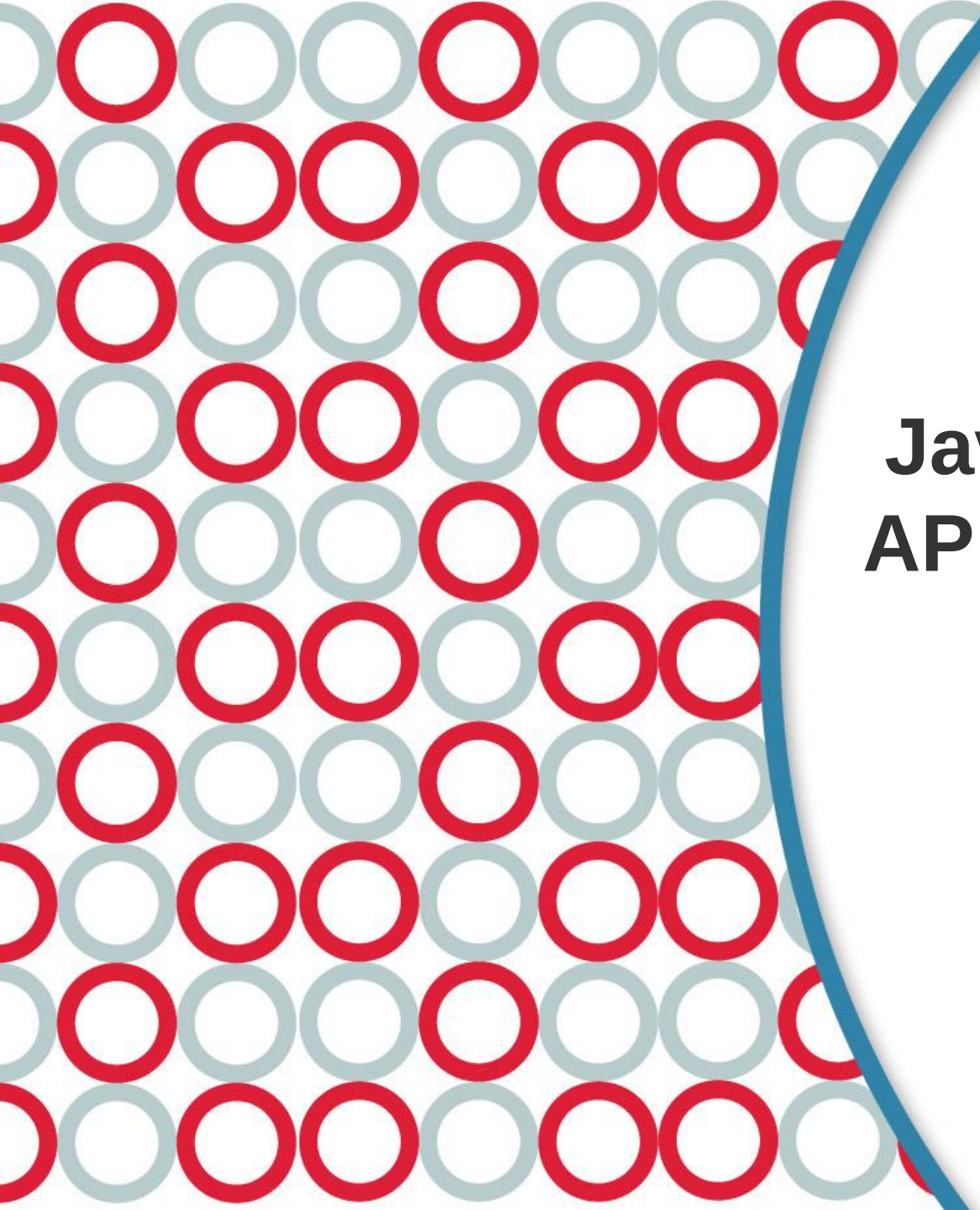


June 2021, IPT Course
Introduction to Spring 5

ORM. Java Persistence API. Transactions

Trayan Iliev

tiliev@iproduct.org
http://iproduct.org

Transactions and Concurrency

- **Transaction = Business Event**
- **ACID rules:**
- **Atomicity** – the whole transaction is completed (commit) or no part is completed at all (rollback).
- **Consistency** – transaction should presetve existing integrity constraints
- **Isolation** – two uncompleted transactions can not interact
- **Durability** – successfully completed transactions can not be rolled back

JPA Transaction Management

❖Global transactions – enable you to work with multiple transactional resources, typically relational databases and message queues (JTA UserTransaction, JNDI lookup).

❖Local transactions – resource-specific, such as a transaction associated with a JDBC connection, but cannot work across multiple transactional resources.

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

Transaction Isolation Levels

- DEFAULT - use the default isolation level of the underlying datastore
- READ_UNCOMMITTED – dirty reads, non-repeatable reads and phantom reads can occur
- READ_COMMITTED – prevents dirty reads; non-repeatable reads and phantom reads can occur
- REPEATABLE_READ – prevents dirty reads and non-repeatable reads; phantom reads can occur
- SERIALIZABLE – prevents dirty reads, non-repeatable reads and phantom reads

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

```java
@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();
… }
```

```java
@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

```java
@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<>();
… }
```

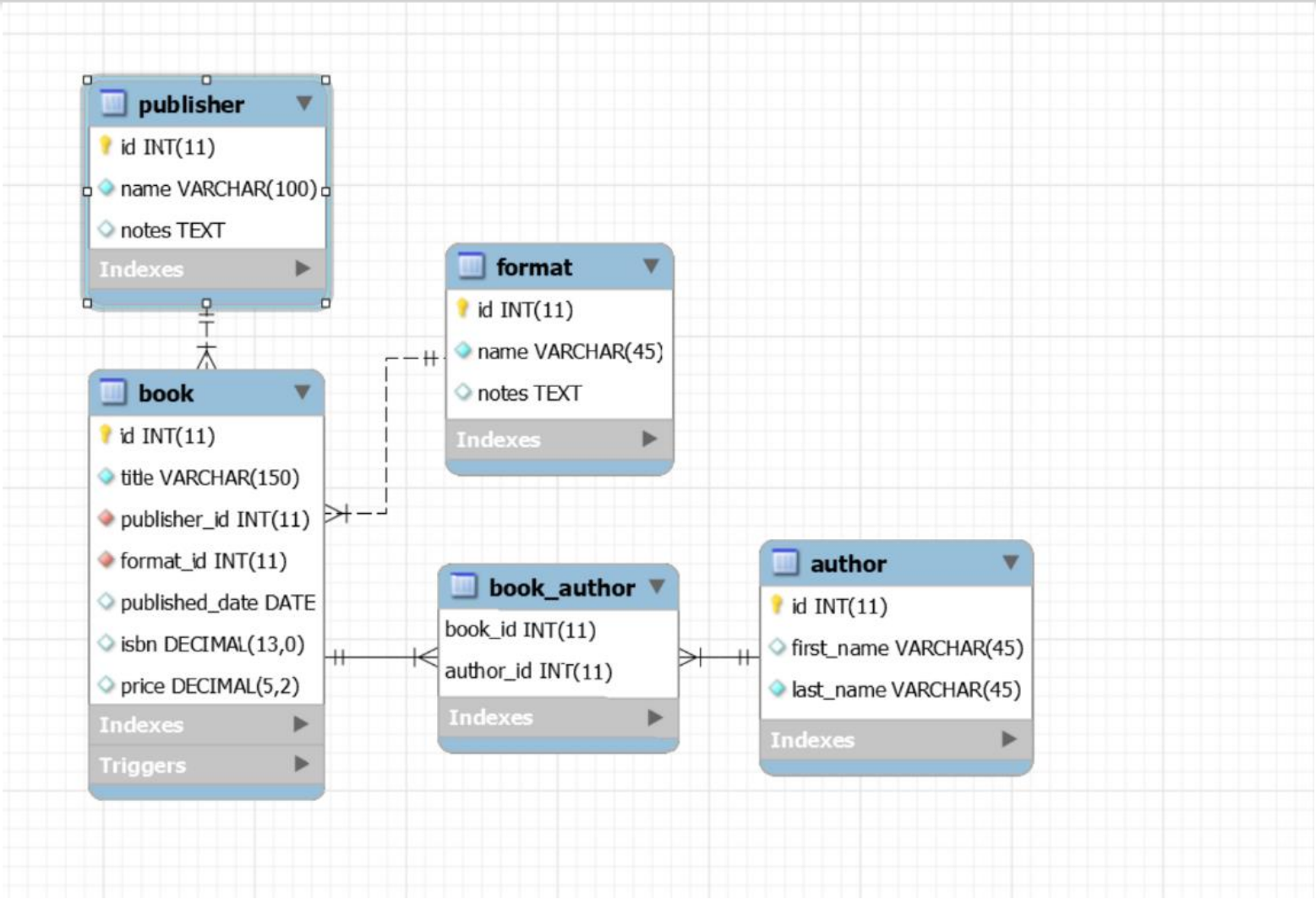
```java
@Entity
public class Author {
      @Id @GeneratedValue
      private int id;

      @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

JSR-303: Bean Validation (1)

- Bean Validation стартира през юли 2006 - JSR 303
- Финализирана е на 16 ноември 2009
- Валидацията е обща задача, която се осъществява през всички слоеве на приложението – от презентационния до персистирането на данните
- Често една и съща логика за валидация се реализира многократно във всеки слой, което води до чести грешки е несъответствия, както и до дублиране на усилия
- За да се справят с проблема, често разработчиците кодират валидационната логика директно в домейн модела, което води до смесване на бизнес логика и метаданни за валидиране на отделните свойства

JSR-303: Bean Validation (2)

- JSR 303: Bean Validation предлага набор от стандартни ограничения (constraints) относно данните, под формата на анотации, които обозначават полета, методи или класове на JavaBean компоненти, като например JPA Entities или JSF Managed Beans
- Има множество предварително дефинирани анотации, както и възможност за създаване на собствени такива и свързването им с клас, който да реализира валидационната логика
- Вградените анотации са дефинирани в пакет javax.validation.constraints

Bean Validation Annotations (1):

- @AssertFalse – елемент от булев тип трябва да е лъжа
- @AssertTrue – елемент от булев тип трябва да е истина
- @Min, @DecimalMin – минимална стойност на елемент от числов тип
- @Max, @DecimalMax – максимална стойност на елемент от числов тип
- @Digits – атрибути fraction и integer за дробната и цялата част на елемент от числов тип
- @Future – валидиране на бъдеща дата (Date и Calendar)
- @Past – валидиране на минала дата (Date и Calendar)
- @Size – min и max размер на String, Collection, Map или Array

Bean Validation Annotations (2):

- @NotNull – елементът трябва да е различен от null
- @Null – елементът трябва е null
- @Pattern – елементът трябва да съответствува на посочения в атрибута regexp регулярен израз
- @Valid – анотация в пакета javax.validation, която указва, че трябва да се извърши рекурсивна валидация на всички обекти свързани с посочения обект
- Възможно е създаване на нови собствени анотации и композитни анотации с използване на @Constraint, @GroupSequence, @ReportAsSingleViolation, @OverridesAttribute

Bean Validation Examples:

```
public class Email {
    @NotEmpty @Pattern(".+@.+\\.[a-z]+")
    private String from;
    @NotEmpty @Pattern(".+@.+\\.[a-z]+")
    private String to;
    @NotEmpty
    private String subject;
    @Min(1) @Max(10)
    private Integer priority;
    @NotEmpty
    private String body;
    ...
}
```

Bean Validation – Custom Annotation:

```
@Size(min=4, max=4)
@ConstraintValidator(validatedBy = PostCodeValidator.class)
@Documented
@Target({ANNOTATION_TYPE, METHOD, FIELD})
@Retention(RUNTIME)
public @interface PostCode {
    public abstract String message() default
            "{package.name.PostCode.message}";
    public abstract Class<?>[] groups() default {};
    public abstract Class<? extends ConstraintPayload>[]
                        payload() default {};
}
```

Bean Validation – Class PostCodeValidator

```
public class PostCodeValidator implements
                                ConstraintValidator<PostCode, String> {
   private final static Pattern POSTCODE_PATTERN =
                                Pattern.compile("\\d{4}");
   public void initialize(PostCode constraintAnnotation) { }
   public boolean isValid(String value,
                                 ConstraintValidatorContext context) {
      return POSTCODE_PATTERN.matcher(value).matches();
   }
}
```

Bean Validation – композитна анотация:

```
@ConstraintValidator(validatedBy = {})   @Documented
@Target({ANNOTATION_TYPE, METHOD, FIELD})
@Retention(RUNTIME)
@Pattern(regexp = "\\d{4}")
@ReportAsSingleViolation
public @interface PostCode {
    public abstract String message() default
            "{package.name.PostCode.message}";
  public abstract Class<?>[] groups() default {};
  public abstract Class<? extends ConstraintPayload>[]
                        payload() default {};
}
```

Additinal Examples

MySQL CREATE INDEX Tutorial – https://www.mysqltutorial.org/mysql-index/mysql-create-index/

MySQL Join Tutorial – https://www.mysqltutorial.org/mysql-join/

MySQL Transaction Tutorial – https://www.mysqltutorial.org/mysql-transaction.aspx

JPA in Java EE 6 Tutorial – https://docs.oracle.com/javaee/6/tutorial/doc/bnbpy.html

Thank's for Your Attention!

Trayan Iliev

CEO of IPT – Intellectual Products & Technologies

http://iproduct.org/

http://robolearn.org/

https://github.com/iproduct

https://twitter.com/trayaniliev

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

https://plus.google.com/+IproductOrg